REALIZATION OF GALOIS SWITCHING FUNCTIONS

A Thesis Submitted
in Partial Fulfilment of the Requirements
for the Degree of
MASTER OF TECHNOLOGY

by

DURGASANKAR BANERJEE

to the

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY, KANPUR

APRIL, 1989

11/4/89.

CERTIFICATE

This is to certify that the work embodied in this thesis titled " REALIZATION OF GALOIS SWITCHING FUNCTIONS " has been carried out by Mr. Durgasankar Banerjee under my supervision and the same has not been submitted elsewhere for a degree.

April - 1989.

(Dr.M. U. SIDDIQI)

Professor
Department of
Electrical Engineering.
Indian Institute of Technology,
Kanpur.

ACKNOWLEDGMENTS

I wish to extend my deep sense of gratitude to Prof. M.U.SIDDIQI for his constant encouragement and patience throughout the thesis work. Specially, the course works offered by him helped me a lot towards the better understanding of my thesis work.

I am very much grateful to Dr. A.Joshi,Dr. V.Sinha,Dr. P.K.Chatterjee,Dr. V.P.Sinha and all of my respected teachers who taught and helped me with their utmost and sincere efforts.

Finally, a lot of thanks are due to my friends Khadilker,Samit,Rampa,Kaveti,Suneel,Vemuri,Sandhu,Usman,Shesadri, Deepak Murthy,Capt. Swamy and many of my well wishers. I convey my many many thanks to dear Venkatesh for his endless and sincere efforts for which only I was able to produce my nice diagrams in ACAD. Daily morning tea session with Kushal Nandy will really be everlasting in my memory. I will always remember my happy days in Hall-4 staying with my near and dear friends specially dada, Subrata Saha, Ramen,Manab,Nayan,Dawn and so many..

April 19th,1989 Durgasankar Banerjee.

Table of contents

ABSTRACT

Galois switching functions (GSFs) can be considered as functions in a vector space with domain $GF(2^m)$ and range $GF(2^n)$. Binary switching functions are a special case of the GSFs. These functions are amenable to a compact and well-structured polynomial representation on an appropriate extended Galois field. This systematic algebraic representation in the form of a polynomial holds for both binary as well as nonbinary switching functions. In this thesis, we have presented an approach to the realization of GSFs. The realization has been done in two stages. In the first stage, we have determined the coefficients of polynomial representation of binary switching functions in software. The second stage consists of polynomial evaluation over a finite field in hardware.

The polynomial representation is obtained for both 1-D and 2-D cases. A truth table of upto a maximum of 15 input and output Boolean variables has been represented in the form of a polynomial of one variable in an extended Galois field for the 1-D case. The representation gives the corresponding coefficients to the truth table in an appropriate extended Galois field. For general mapping with the domain $\in GF(2^m)$ and range $\in GF(2^n)$, appropriate zero padding reduces the field size which improves the speed of computation. For 2-D GSFs, 2-D images are represented in the form of polynomials of 2-variable in an extended Galois field.

Various algorithms have been explained with the help of flow charts. The software package is modular and user interactive.

Hardware to obtain the original mapping has been developed. This consists of polynomial evaluation based on Horner's rule over $GF(2^N)$ with N in the range of 2 to 8. For this, we have developed a finite field multiplier over $GF(2^N)$ for any N in the above mentioned range using standard basis.

CHAPTER 1

INTRODUCTION

Galois switching functions (GSFs) are essentially signals defined on a finite index set with the structure of a finite field, say $GF(2^k)$, and taking their values from a finite field, say $GF(2^n)$. The GSFs may be interpreted as members of a vector space with domain $GF(2^k)$ and range $GF(2^n)$. These are a generalization of binary switching functions in that the input and output variables can assume values over any finite field.

Galois switching functions can be very useful in any digital information processing situation such as error control coding, cryptography, and digital image processing.

Digital information processing is essentially centered around various suitable representations of a set of n Boolean functions of m variables. These representations should be simple and systematic for information processing purposes. If the Boolean functions are represented in a simple algebraic form, further processing will involve only algebraic computations. Several authors, as referred to the next section, have contributed towards formulation of algebraic representations of Boolean functions.

Representation of signals for digital information processing purposes calls for an alternative description of the following set of Boolean functions

$$y_j = f_j(x_0, x_1, \ldots\ldots, x_{m-1}),$$

where $j = 0, 1, 2, \ldots\ldots, n-1$ and $x_i, y_j \in \{ 0, 1 \}$. Since Galois or finite field is the natural extension of the Boolean field, the concept of finite fields can be utilized for representing Boolean functions in polynomial form. Polynomial representation in an extended Galois field realizes a Boolean function mapping. In applications such as error control coding and cryptography, Boolean functions can be realized systematically in polynomial form. Application of Galois switching functions also plays an important role in case of nonbinary switching functions.

In this thesis, we have considered the representation of the Boolean functions in polynomial form over a Galois field. As the Boolean functions are represented by a polynomial, further processing will be in the form of algebraic operations on the polynomial.

We have developed the software to realize polynomial coefficients representing a given Galois switching functions for both 1-D and 2-D cases. For a 1-D Galois switching function, we have represented in polynomial form a truth table for upto a maximum of 15 Boolean functions, each one being a function of upto 15 variables. Evaluation of the coefficients involves DFT computation over finite field, for which FFT algorithms over finite fields have been implemented. For realization of 2-D Galois switching functions, we have implemented 2-variable polynomial representations of images. Grey

levels of 2-D images have been expressed as field elements in $GF(2^n)$ and the image data has been realized in a polynomial form of 2 variables in an extended Galois field. Evaluation of polynomials over Galois fields is an integral part of realization of the original mapping. For this we have presented an approach to design a hardware for evaluation of a single variable polynomial in $GF(2^N)$, for N lying between 2 and 8. Polynomial evaluation is based on Horner's rule.

1.1 Historical background:

Several authors have presented their discussions on switching functions based on a Galois field. Ninomiya [9] was the person who first treated the realization of switching functions in a Galois field. Bartee, Schneider [1], Benjauthrit [2] and Reed [3] introduced the idea of use of extension fields. Menger [8] presented the viewpoint of the abstract Fourier transform. Pradhan and Patel [10] proposed the idea of minimization of switching functions based on the Reed-Muller code which is basically a multivariable polynomial in a Galois field. Recently, Takahashi [11] proposed a method to reduce n functions of m variables on GF(2) to one polynomial of one variable in the extended Galois field $GF(2^N)$. He also pointed out some remarkable properties of such polynomials based on Frobenius transforms.

1.2 Organization of the thesis:

In this section we give a brief account of the remaining chapters in the thesis.

Chapter 2 introduces the relevant algebraic background concerning finite fields. An outline of some basic algorithms for finite field fast Fourier transform (FFT) is also given here.

In Chapter 3, we have presented the theory of polynomial representation of switching functions in extended Galois fields. Related theorems have been stated.

Chapter 4 gives a software package for determining polynomial coefficients representing a given Galois switching function. Flow charts of various algorithms implemented in the software have been given. Determination of polynomial coefficients has been done for the both 1-D and 2-D Galois switching functions. For 1-D Galois switching functions, we have implemented the polynomial representation of truth tables of a maximum of upto 15 Boolean functions each of 15 variables. As an application of such 1-D polynomial representations, we have implemented the substitution mapping realization of upto 8 objects. For 2-D Galois switching functions, we have determined the coefficients for the polynomial representation of 2-D images.

Chapter 5 gives details of the hardware implementation. The hardware implements the polynomial evaluation based on Horner's rule. We have presented a multiplier design to

realize multiplications over a Galois field GF(2^N), where N is in the range of 2 to 8. Detailed circuit and timing diagrams are given.

Results and future scope have been discussed in Chapter 6 as the concluding part of the thesis.

CHAPTER 2

ALGEBRAIC BACKGROUND

This chapter includes relevant details of Galois fields. Fast fourier transform over finite field has also been discussed.

2.1 Finite fields and extensions:

In this section we have given the definitions of finite field, extension of finite field and other necessary terms.

2.1.1 Finite fields:

A field is a set containing atleast two elements that has two operations defined on it - addition and multiplication - such that the following axioms are satisfied:

a) The set is an abelian group under addition (+).

b) The set is closed under multiplication (*), and the set of nonzero elements is an abelian group under multiplication.

c) The distribution law

$(a + b)c = ac + bc$ holds for all a,b,c in the set.

Some of the examples of fields are the set of real numbers, the set of complex numbers, the set of rational numbers. The above fields have an infinite number of elements known as infinite field. A finite field is one which contains a finite number of elements. A finite field containing q elements is

known as Galois field, denoted by GF(q). Every field must have an element zero and an element one. The field containing only these two elements is known as GF(2). The addition and multiplication tables are given below for GF(2):

+	0	1
0	0	1
1	1	0

.	0	1
0	0	0
1	0	1

For a finite field GF(q), q must be either a prime number, or some integral power of a prime number, i.e., $q = p^n$, $n = 1, 2, 3, ..$ and p is a prime number.

2.1.2 Field extension:

Let F be a field. A subset K of F that is itself a field under the operations of F is called a subfield of F. In this context, F is called an extension field of K. If K = F, we say that K is a proper subfield of F. As for example, the field of rationals is a subfield of the field of reals, which in turn is a subfield of the complex field. There may be only a finite number of integers in the field, in which case the number of integers is called the characteristic of the field. The characteristic of the field is always a prime number. As for example, characteristic of GF(3) is 3, GF(2) is 2. Both the real and complex fields have characteristic infinity.

2.2 Polynomial rings:

For each field F, there is a ring F[x] called the ring of polynomials over F. Mathematically, a polynomial over a field is: $f(x) = f_n x^n + f_{n-1}x^{n-1} + \ldots\ldots + f_1 x + f_0$.

$$= \sum_{i=0}^{n} f_i x^i .$$

where the coefficients $f_0, f_1, f_2, \ldots, f_n$ are elements of the field. The zero polynomial is $f(x) = 0$. A monic polynomial is a polynomial whose coefficient f_n with the largest index is equal to 1. The sum of two polynomials in F[x] is another polynomial in F[x] defined by

$$f(x) + g(x) = \sum_{i=0}^{n} (f_i + g_i)x^i .$$

The degree of the sum is not greater than the larger of these two degrees. The product of two polynomials in F[x] is another polynomial in F[x] defined by

$$f(x)g(x) = \sum_{i=0} \left(\sum_{j=0} f_j g_{i-j} \right) x^i .$$

The degree of product is equal to the sum of the degrees of the two polynomials. A nonzero polynomial p(x) which is only divisible by p(x) or β, where β is an arbitrary field element, is called an irreducible polynomial. A monic irreducible polynomial is called a prime polynomial.

2.3 Representation of field elements by power of primitive element:

An element of a field GF(q) is said to be a primitive element which can generate all the other nonzero field elements. The order of a field element is defined as the least integer, to which the element should be raised to get the unit element of the field. The order of a primitive element of GF(q) is (q-1),i.e., $\beta^{q-1} = 1$, where β is the primitive element of GF(q). Let us present an example for the representation of field elements in $GF(2^3)$ as the power of primitive element β.

Example 2.3.1: Let $p(x) = x^3 + x + 1$ be an irreducible polynomial over GF(2) and β be the primitive element in $GF(2^3)$. Now, $\beta^3 + \beta + 1 = 0$. Here in below both the n-tuple i.e cartesian form and the polynomial form are given for $GF(2^3)$. Polar form is the representation of any field element in the power of primitive element.

n-tuple	polynomial representation	polar form
0 0 0	0	0
0 0 1	1	$1 = \beta^0$
0 1 0	x	β
1 0 0	x^2	β^2
0 1 1	$1 + x$	β^3
1 1 0	$x + x^2$	β^4
1 1 1	$1 + x + x^2$	β^5
1 0 1	$1 + x^2$	β^6

2.4 Vector spaces:

For a given field F, the n-tuple of field elements $(f_0, f_1, \ldots\ldots, f_{n-1})$ is called a vector of length n over the field F. The set of all such vectors of length n together with two operations called vector addition and scaler multiplication is called a vector space over the field F. Scaler multiplication is an operation that multiplies a vector by a field element. Symbolically, $\beta(f_0, f_1, f_2, \ldots, f_{n-1}) = (\beta f_0, \beta f_1, \beta f_2, \ldots, \beta f_{n-1})$ where, β is a field element and $(f_0, f_1, f_2, \ldots, f_{n-1})$ are the components of a vector $\underline{f}$.

Vector addition is an operation that adds two vectors $\underline{g}$ and $\underline{h}$ according to the following definition:

$$(g_0, g_1, \ldots\ldots, g_{n-1}) + (h_0, h_1, \ldots\ldots, h_{n-1})$$
$$= (g_0 + h_0, g_1 + h_1, \ldots\ldots, g_{n-1} + h_{n-1}).$$

For a vector space V, the operations must satisfy the following axioms:

a) V is an abelian group under the vector addition.

b) For any vectors V1, V2 and any scaler β

$$\beta(V1 + V2) = \beta V1 + \beta V2.$$

For any vector V and any scalers β and π

$$(\beta + \pi) V = \beta V + \pi V.$$

These are known as distributive laws.

c) For any vector V and any scalers β and π

$$(\beta\pi) V = \beta(\pi V).$$

This is known as the associative law.

2.5 Standard and normal basis:

For a vector space we can construct basis consisting of linearly independent vectors. We can then represent any vector in the vector space in terms of the basis. The vector space for $GF(2^n)$ can be represented by n linearly independent vectors which constitute a basis. In Galois field there are basically two types of representation. One is the standard basis and the other is normal basis.

a) Standard basis : If α is a primitive element in $GF(2^n)$, then the set $\{ 1, \alpha, \alpha^2, \alpha^3, ..., \alpha^{2^{n-1}} \}$ forms a basis vector in the vector space formed by $GF(2^n)$. This is known as standard basis. We have used this representation throughout our thesis work.

b) Normal basis : If α is a primitive element in $GF(2^n)$, we can find a β such that $\beta = \alpha^k$ where k and n are the positive integers. The set $\{ \beta^{2^0}, \beta^{2^1}, \beta^{2^2}, ..., \beta^{2^{n-1}} \}$ constitutes the basis of the vector space formed by $GF(2^n)$ known as normal basis. For a given field $GF(2^n)$, we can have many normal bases each having the same number of elements. The only difference is in the relation between the polar and the corresponding cartesian representation.

2.6 Fast Fourier transform over finite field:

One of the important computations in digital signal processing is discrete Fourier transform. There is also a Fourier transform in a finite field. Discrete Fourier transform in a

finite field F is defined as:

$$V_k = \sum_{i=0}^{n-1} w^{ik} v_i \qquad \ldots\ldots\ldots(2.1)$$

$$k = 0, 1, 2, \ldots, n-1$$

where, w is an nth root of unity in F and v and V are the vectors of length n in F. The above equation requires in the order of n^2 multiplications and n^2 additions. Using FFT algorithms we need in the order of nlogn operations. A Fourier transform of block length n exists in F if there is an nth root of unity in F. Say $v_i \in GF(q)$ and $V_j \in GF(q^n)$, then the nth root of unity in F will exist if n divides $(q^m - 1)$.

A 2-D discrete Fourier transform in a field F is defined as:

$$V_{m,n} = \sum_{i=0}^{n1-1} \sum_{j=0}^{n2-1} w^{im} \pi^{jn} v_{i,j}$$

where m = 0, 1, ..., n1-1 and n = 0, 1, ..., n2-1.

This transform exists whenever the field F contains an element w of order n1 and an element π of order n2. If such elements do not exist, then the transform is not possible. There are many algorithms [4] available on fast Fourier transform like Cooley-Tukey, Good Thomas, Rader prime etc. We shall discuss in breif those algorithms which have been implemented in the software. Proofs of the algorithms [4] have not given here.

2.6.1 Cooley-Tukey algorithm:

The Fourier transform of a vector v is given by the equation (2.1). If the block length n is composite say n = n1.n2, then the summarised structure of Cooley-Tukey algorithm [4] is given below.

$$n = n1.n2$$

$$i = i1 + n1.i2$$

$$k = n2.k1 + k2$$

Then $V_{k1,k2} = \sum_{i1=0}^{n1-1} \beta^{i1.k1} \left[w^{i1.k2} \sum_{i2=0}^{n2-1} \pi^{i2.k2} v_{i1,i2} \right]$,

where $i1 = 0, 1, \ldots, n1-1$,

$i2 = 0, 1, \ldots, n2-1$,

$k1 = 0, 1, \ldots, n1-1$,

$k2 = 0, 1, \ldots, n2-1$,

$w^n = 1$, $w^{n1} = \pi$ and $w^{n2} = \beta$.

Number of multiplications is in the order of n.(n1 + n2) + n.

2.6.2 Rader prime algorithm:

The Rader prime algorithm [4] can be used to compute the Fourier transform in any field F for a prime length. For a prime length p, we can construct a field GF(p) to reindex the vector components. By this algorithm n point Fourier transform can be converted into (n -1) points cyclic convolution. Let π be the primitive element in GF(p). Then each integer less than p can be expressed as a unique power of π.

Now, the discrete Fourier transform is:

$$V_k = \Sigma\, w^{ik} v_i \qquad k=0,\ 1,\ .....,p-1$$

This can be written as,

$$V_0 = \sum_{i=0}^{n-1} v_i$$

$$\text{and} \quad V_k = v_0 + \sum_{i=1}^{n-1} w^{ik} v_i \qquad k = 1,2,....,n-1$$

For each i, let r(i) be the unique integer from 1 to (n-1) such that in GF(p), $\pi^{r(i)} = i$. The function r(i) is a map from the set { 1, 2, ...,n-1 } onto the set { 1, 2, ...,n-1 }. Then we can write V_k as,

$$V_{\pi r(k)} = v_0 + \sum_{i=1}^{n-1} w^{\pi r(i)+r(k)} \cdot v_{\pi r(i)}$$

Because r(i) is a permutation, we can put L = r(k), j = n -1-r(i).

$$V_{\pi L} = v_0 + \sum_{j=0}^{n-1} w^{\pi L-j} v_{\pi n-1-j}$$

This can be written as:

$$V_L^{(1)} = v_0 + \sum_{j=0}^{n-2} w^{\pi L-j} v_j^{(1)} \qquad .....(2.2)$$

where $V_{\pi L}^{(1)} = V_L^{(1)}$ and $v_j^{(1)} = v_{\pi n-1-j}$ are the scrambled input and output data sequences. From the equation (2.2), we see that the n point Fourier transform is converted into (n-1) point cyclic convolution. We can realise FFT by combining Rader prime algorithm and Winograd convolution algorithm.

CHAPTER 3

THEORY OF POLYNOMIAL REPRESENTATION

3.1 Introduction:

Representation of `n' functions of `m' variables on GF(2) can be done by one polynomial of one variable on the extended Galois field $GF(2^n)$. We shall discuss the theory of this polynomial representation in the following section.

3.1.1 Representation of simple 1-D mapping :

Let us first consider the simple case of a truth table for which m = n, where m is the number of input variables and n is the number of output variables. Then $x = (x_0\ x_1\ x_2\ \ldots\ x_{n-1})$ and $y = (y_0\ y_1\ y_2\ \ldots\ y_{n-1})$ can be represented as elements of $GF(2^n)$, where $GF(2^n)$ is the extended Galois field over GF(2). The polynomial representation of such a simple truth table is:

$$y = f(x). \qquad x,y \in GF(2^n).$$

Theorem [11] associated with this polynomial representation is given below.

Theorem: Any function f(x) on $GF(2^n)$ can be represented as a polynomial of order $r = 2^n - 1$; i.e.,

$$f(x) = a_0 + a_1x + a_2x^2 + \ldots + a_rx^r,\ a_i \in GF(2^n).$$

The coefficients a_i ($0 \leq i \leq r$) are determined by

$$a_0 = f(0),$$

$$a_i = \sum_{x \in GF(2^n)} x^{r-i} f(x), \quad 1 \le i \le r, \quad x \in GF(2^n)$$

Proof of this theorem is given in Appendix A.

3.1.2 Representation of 1-D general mapping :

In general m is not necessarily equal to n. In this case the theorem for its polynomial representation is given below:

Theorem: Any function f(x) can be represented as a polynomial

$$f(x) = a_0 + a_1 x + a_2 x^2 + \ldots\ldots + a_r x^r \qquad \ldots\ldots (3.2)$$

$r = 2^m - 1$, $x \in GF(2^m)$, $f(x) \in GF(2^n)$, $a_i \in GF(2^L)$ $(0 \le i \le r)$

where coefficients a_i $(i = 0,..,r)$ are determined by

$$a_0 = f(0),$$

$$a_i = \sum_{x \in GF(2^m)} x^{r-i} f(x), \quad 1 \le i \le r. \qquad \ldots\ldots (3.3)$$

And L is the l.c.m of m and n.

For the general mapping $x \in GF(2^m)$ and $y \in GF(2^n)$. The set of all such mappings has the structure of a vector space of dimension 2^m. In this vector space we can choose 2^m basis elements shown in the next page.

x	x^0	x^1	x^2	x^3	.....	x^{2^m-1}
0	1	0	0	0	.....	0
1	1	1	1	1	.....	1
β	1	β	β^2	β^3	.....	1
β^2	1	β^2	β^4	β^6	.....	1
β^3	1	β^3	β^6	β^9	.....	1
.	.	.	.	.	.....	1
β^{2^m-2}	1	β^{2^m-2}	. .		.....	1

Here, $x^{2^m} = x$; $x \in GF(2^m)$. The set of all such special functions (x^0 x^{2^m-1} x^{2^m-2} ..... x^2 x) are linearly independent and constitute a basis of the vector space under consideration. Now $y = f(x)$ can be written as

$$y = a_{\beta^{-\infty}} + a_{\beta^0} x^{2^m-1} + a_{\beta^1} x^{2^m-2} + \ldots + a_{\beta^{2^m-2}} x$$

In matrix notation , $\underline{f} = G \; \underline{a}$ ..... (3.4)

where the matrix G can be written as

$$\begin{bmatrix} 1 & 0 & 0 & \ldots & 0 & 0 \\ 1 & & & & & \\ 1 & & & & & \\ 1 & \multicolumn{4}{l}{\text{DFT matrix of}} & \\ 1 & & & & & \\ . & \multicolumn{4}{l}{\text{order } 2^m-1} & \\ . & & & & & \\ 1 & & & & & \end{bmatrix}_{2^m \times 2^m}$$

The inner matrix of G is a Dft matrix of order (2^m-1).

The equation (3.4) gives the forward Galois transform of f.

Now we can also write, $\underset{-}{a} = G^{-1}\underset{-}{f}$ .....(3.5)

where the inverse matrix of G is

$$\begin{bmatrix} 1 & 0 & 0 & \ldots & 0 & 0 \\ 1 & & & & & \\ 0 & & \text{IDFT matrix} & & & \\ 0 & & & & & \\ 0 & & \text{of order} & & & \\ 0 & & & & & \\ 0 & & 2^m - 1 & & & \\ . & & & & & \\ . & & & & & \\ 0 & & & & & \end{bmatrix}$$

The inner matrix of G^{-1} is an IDFT matrix of order $(2^m - 1)$.

The equation (3.5) gives the reverse Galois transform of f. From this equation we can find out `a' coefficients as,

$$a_{\beta^{-\infty}} = f_{\beta^{-\infty}} \qquad \ldots\ldots\ldots(3.5a)$$

$$a_{\beta^0} = f_{\beta^{-\infty}} + f_{\beta^0} + f_{\beta^1} + \ldots\ldots\ldots + f_{\beta^{2^m-2}}.$$

And other components of `a' coefficients can be determined by the corresponding IDFT components. Determining all those coefficients we can get the polynomial representation of switching functions.

3.1.3 General theorems on Galois switching functions:

For the realization of truth table in extended Galois field, we have so far discussed earlier the case where the ground field is GF(2). For general Galois field GF(q), where q is a prime number, the relevant theorems [11] are given below.

3.1.3A Single variable polynomial representation:

Any function f(x) on GF(q) can be represented by a polynomial of order r = q - 1 as

$$f(x) = a_0 - a_1x - a_2x^2 - \ldots\ldots\ldots - a_rx^r \quad \ldots\ldots\ldots (3.6)$$

where $a_i \in GF(q)$, $a_0 = f(0)$

and $$a_i = \sum_{x \in GF(q)} x^{r-i} f(x). \quad \ldots\ldots\ldots (3.7)$$

3.1.3B m-variable polynomial representation:

This is the generalized theorem of the above in the case of m-variable functions.

Any m-variable function on GF(q)

$$y = f(x1,x2,x3,\ldots,xm) \quad (x1,x2,x3,\ldots xm,y \in GF(q))$$

can be represented by an m-variable polynomial as

$$f(x1,x2,\ldots,xm) = \sum_{i1=0}^{r} \sum_{i2=0}^{r} \ldots \sum_{im=0}^{r} a_{i1i2\ldots im}\, x1^{i1} x2^{i2} \ldots xm^{im} \quad \ldots (3.8)$$

$$(r = q - 1,\ a_{i1i2..im} \in GF(q)),$$

where

$$a_{00....0} = f(0,0,...,0),$$

$$a_{i0....0} = -\sum_{x} x^{r-i} f(x,0,...,0),$$

$$a_{0i....0} = -\sum_{x} x^{r-i} f(0,x,....,0), \qquad .....(3.9)$$

$$.....$$

$$a_{00....i} = -\sum_{x} x^{r-i} f(0,0,....,x),$$

$$a_{ij0....0} = (-1)^2 \sum_{x}\sum_{y} x^{r-i} y^{r-j} f(x,y,0,..,0)$$

$$a_{i0j0...0} = (-1)^2 \sum_{x}\sum_{y} x^{r-i} y^{r-j} f(x,0,y,0,..,0)$$

$$.....$$

$$a_{00...0ij} = (-1)^2 \sum_{x}\sum_{y} x^{r-i} y^{r-j} f(0,0,...,0,x,y)$$

$$.....$$

$$a_{i1i2...im} = (-1)^m \sum_{x1}\sum_{x2}..\sum_{xm} x1^{r-i1} x2^{r-i2} ...xm^{r-im} f(x1,x2,..,xm)$$

$$1 \leq i1,i2,i3,...,im \leq r.$$

where in $\sum_{x}$, $\sum_{x}\sum_{y}$ ...., x,y ... run all over GF(q).

3.1.4 Frobenius cycles in switching functions:

The polynomial y = f(x) over finite field as

discussed earlier has nice properties connected with Frobenius cycles [11]. This is based on the following theorem about finite fields.

Theorem: Let $P = GF(q^p)$ be an extension field of $K = GF(q)$ where q is a prime power. Any $\theta \in P$ is in K if and only if

$$\theta^q = \theta.$$

In general when a finite field P has a subfield K ($\subseteq P$), the transformation $\theta \rightarrow \theta^q$ for any $\theta \in P$ is called 'K-Frobenius transformation.'

From the above theorem, $\theta \in P$ is invariant by 'K-Frobenius transformation' iff $\theta \in K$. $\{ \theta, \theta^q, \theta^{q^2}, \ldots, \theta^{q^{i-1}} \}$ is called a K-Frobenius cycle if $\theta^{q^i} = \theta$ and $\theta^{q^j} \neq \theta$ for $j < i$. Sum of all elements of a K-Frobenius cycle is known as 'trace' of θ, i.e, $tr(\theta) = \theta + \theta^q + \theta^{q^2} + \ldots + \theta^{q^{i-1}}$.

Another important theorem based on the polynomial representation is given below.

Theorem: Let $GF(2^n) = GF(q) = K$, then coefficients $a_0, a_1, \ldots a_r$ in (3.7) satisfy the following condition

$$a_j = a_i^q$$
$$j = iq \ (\mathrm{mod}(2^m-1)).$$

By this theorem we can say that if $a_i x^i$ is a term in (3.2), then its GF(q)-Frobenius transform $a_i^q x^{iq}$ can also be found in (3.2) ($q=2^n$). As a result terms in (3.2) can be decomposed into several GF(q)-Frobenius cycles and each of the trace functions can be processed in parallel to get higher throughput.

CHAPTER 4

SOFTWARE DEVELOPMENT

In this chapter we discuss in detail the software implementation to determine the polynomial coefficients of Galois switching functions.

4.1 Specification:

Let us first establish the specifications of the problem implemented in this software.

Our problem is to obtain polynomial representation of a given truth table of `m' input and `n' output variables in an extended Galois field. We restrict the maximum field size to $GF(2^{15})$. The range of values of `m' and `n' becomes $2 \leq m,n \leq 15$. User is asked to enter the values of `m' and `n'. Then the entries of X and Y of the truth table will be entered by the user from keyboard. After each entry of X and Y, the user has the option to enter the next entry or to quit the entry so that rest of the entries for Y are taken as don't care. After realization of the polynomial representation of the given truth table, the computed `a coefficients' will be displayed on the screen. The representation can be checked by entering X and the corresponding Y will be computed with those `a coefficients'. This computed Y is the same as in the truth table.

4.2 Programming language:

The main aim of the software realization is to minimize the computational time. This will essentially depend on the implementation of efficient algorithms. In addition to the algorithms, computational time will depend on the programming language used. The software has been developed in 8086 assembly language. Using this assembly language rather than any high level language, the computational time has been reduced a lot. As for example, the realization of truth table for m=8 and n=8 by Fortran IV programme takes time for computation of `a coefficients' as one and half minute. The corresponding time using the assembly language is 2.5 seconds.

4.3 Development and implementation of various algorithms:

Let us discuss various algorithms in detail which have been implemented in the software.

4.3.1 Algorithm for generation of field elements of GF(2^k):

Let α be a primitive element of the extended Galois field GF(2^k) with a primitive polynomial over GF(2). Any field element can be represented as a power of the primitive element. This representation is known as `Polar form'. Another representation is known as `Cartesian form' in which any field element can be represented as a polynomial of α of degree (k-1)

with the coefficients from GF(2), that is

$$\alpha^j = z(0)+z(1).\alpha+z(2).\alpha^2+.....+z(k-1).\alpha^{(k-1)}.$$

where $z(i) \in GF(2)$, $0 \leq j \leq 2^k-2$ and $0 \leq i \leq (k-1)$. Coefficients $z(0)$, $z(1),z(2),....,z(k-1)$ represent α^j in cartesian form. The flow chart for the implemented algorithm to generate the field elements of $GF(2^k)$ in cartesian and polar form is shown in Fig. 4.1. `Primitive poly.' is the primitive polynomial of degree (k-1) with the kth bit as zero. Cartesian form is in k bit register with the corresponding polar form in the `count'. Thus all the cartesian forms are stored in `Pol_buff' corresponding to the address index in polar form. Except the elements 0 and 1 all other field elements are generated by this algorithm. Exponents of field elements 0 and 1 in Polar form are represented as -1 and 0 respectively.

4.3.2 Computations in Galois field:

To increase the speed of operation, the computations in Galois field should be as efficient as possible. This efficiency results from the compromise of speed and memory used. The optimized compromise will fetch the best result.

For implementing any algorithm over Galois fields, we need the efficient arithmetic operations over Galois field. Let us consider the two arithmetic operations ,i.e.,multiplication and

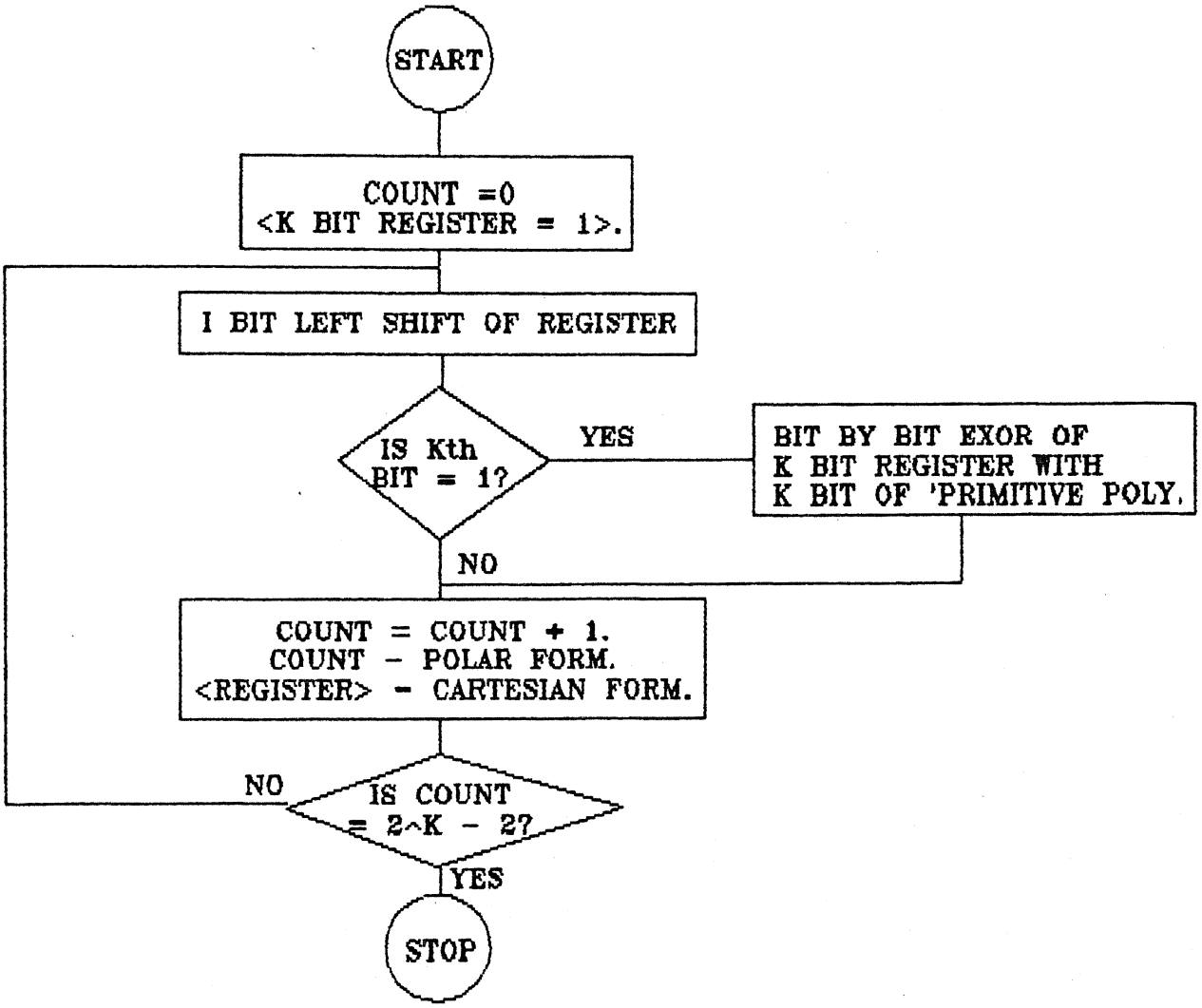


Fig. 4.1 : GENERATION OF FIELD ELEMENTS OF GF(2^K)

addition over GF(2^n). If the elements in the field are represented in polar form, the multiplication of two elements will be just the addition of the powers of primitive element with modulo operation over (2^n -1). For addition of two field elements available in polar form, the direct method is to convert the two elements from polar to cartesian followed by the bit by bit EXOR of those two k-tuples (cartesian form) giving the result in cartesian form only. To get the sum in polar form again we need the conversion from cartesian to polar. Thus the conversion of cartesian to polar and vice versa are often needed. Polar to cartesian conversion can be done efficiently by storing cartesian form of the field elements corresponding to the address index in the polar form in a buffer `Pol_buff'. This needs total of 2^n words for storage. But the conversion of cartesian to polar form becomes difficult. For a smaller value of n say upto n=10 this conversion can be achieved just by brute force comparison method. For large values of n, this will result in poor computational speed. For example, for n=15 the brute force search technique may take 2^{15}=32,768 comparisons in the worst case. Another straight forward method to reduce this time is to construct a buffer of 2^n words which will store the polar form corresponding to the address index in the cartesian form. Though the speed will improve, the memory requirement will be doubled. To implement the optimized algorithm those conversions should be avoided. We shall do all the arithmetic operations in polar form only. By this form, the

multiplication becomes simpler. For addition of two field elements in polar form, it is suitable to use the following relation [11],

$$\theta^i + \theta^k = 1 \ldots\ldots\ldots (4.1)$$

Storing these relations we can add two field elements directly in polar form. We need not store all the relations, but only the relations for each of the first element of conjugacy classes. The amount of memory needed, in general is of the order of $2^n/n$. But with the storing of only the first element of each conjugacy class, for any i determination of k becomes complicated, though memory requirement is considerably reduced. Hence for $n > 16$ we can go through this complication, as the storage needed is considerably large for higher n. For our problem, since n lies between 2 and 15, we need not go for such complexity. Using 2^{n-1} words storage only for the relations given by (4.1), the addition can be done efficiently in the polar form. Thus storing only the first half of the relations, we can find the value of k for any i with a simple logic. The corresponding flowchart is given in Fig. 4.2. To store initially half of the relations we can construct two arrays, one containing polar form with the address index in cartesian and the other containing cartesian form with the address index in polar form. Then after storing the first half of the relations we do not need the two arrays at all. Now we can do the field addition of any two elements θ^x and θ^y in polar form. Flowchart for this field

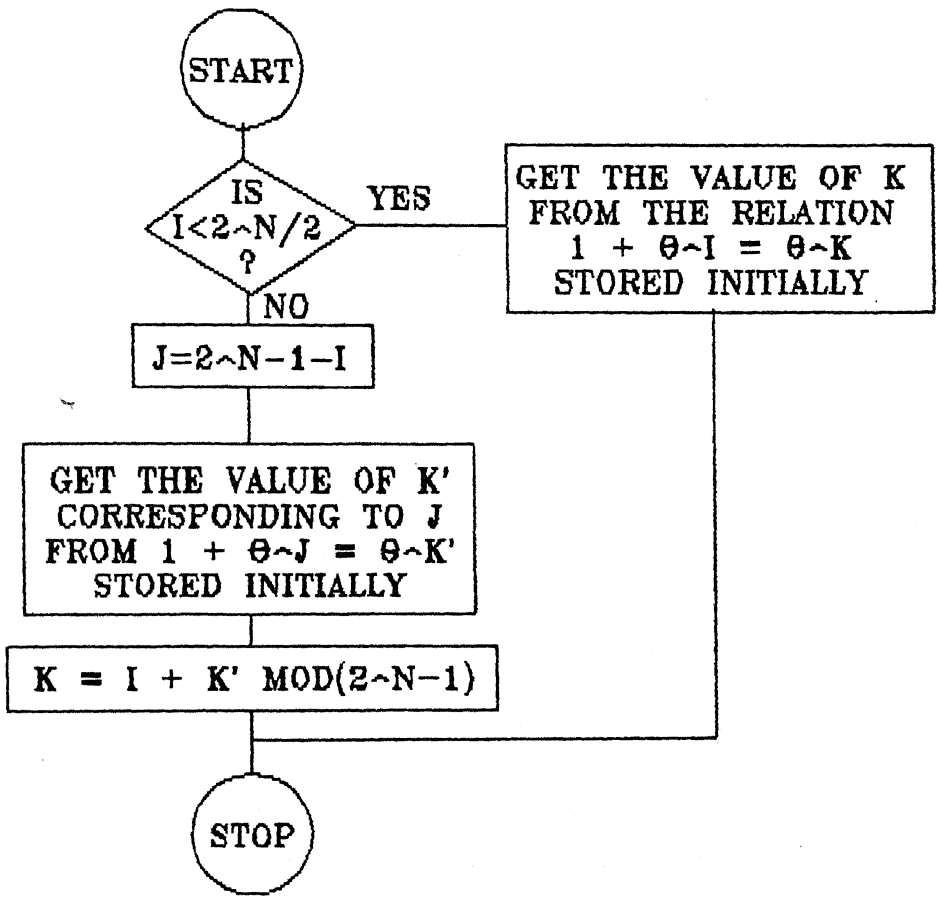


Fig. 4.2 : GENERATION OF `K' FROM `I' BY STORING HALF OF THE RELATIONS 1 + θ^I= θ^K

addition is shown in Fig. 4.3. Result of this addition is also in in polar form. In future all the computations will be done in polar form only.

4.3.3 Zero padding:

We have already stated in Chapter 3 that the coefficients `a' are the elements of $GF(2^L)$ where L is the l.c.m of m and n. Thus some restrictions on m and n should be imposed because of maximum field size as $GF(2^{15})$. As for example, in the case of m=3 and n=7, value of L =21. To get a wider range of values of m and n, it has been proposed to do the zero padding to Y. By this padding concept we raise the value of n-tuple of Y to the nearest multiple value of m-tuple of X. For the above example, we can do two zero paddings to n-tuples of Y. With this padding field size of `a' coefficients becomes $GF(2^9)$ instead of $GF(2^{21})$. Flow chart of this zero padding is given in Fig. 4.4. Even introducing the zero padding to Y, we can not take all the combinations of m and n. As for example, in the case of m=6 and n=13 we can not pad to Y further. Thus for each value of m entered by the user, the maximum value of n possible is displayed on the screen. For the above example, the maximum possible value of n=12 for m=6. After the padding we need to store the relations given by (4.1) for the field $GF(2^n)$.

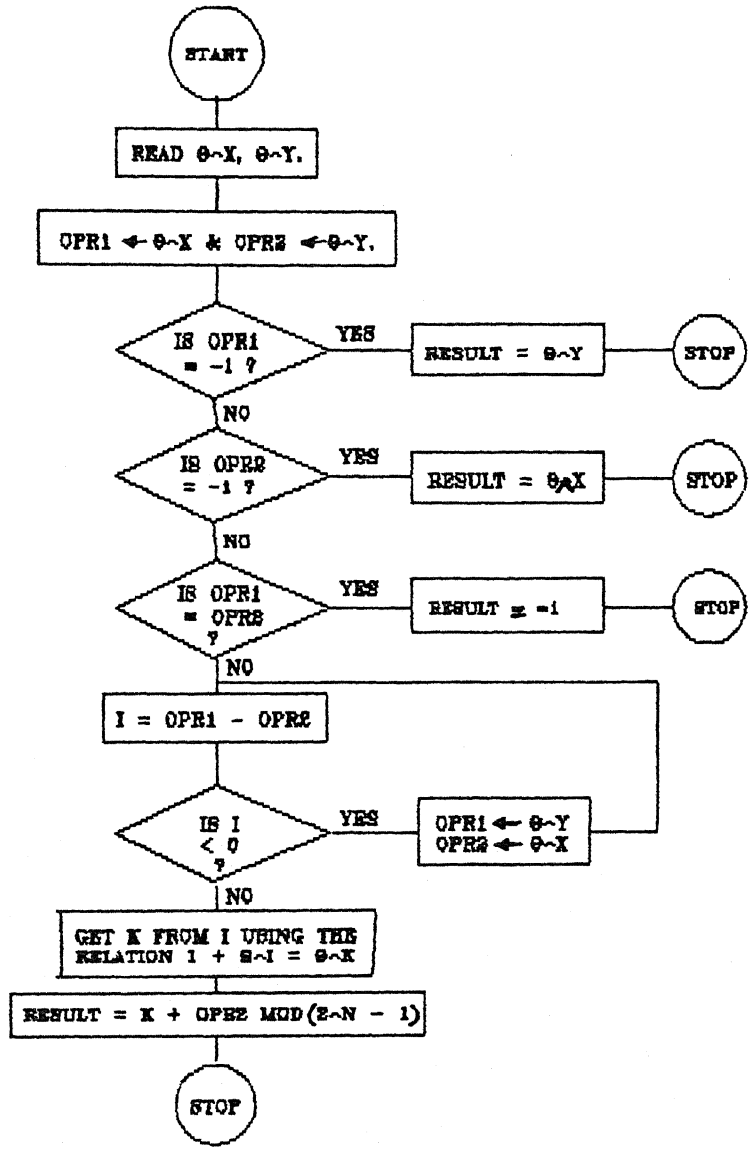


Fig. 4.3 : ADDITION OF θ^X AND θ^Y IN POLAR FORM

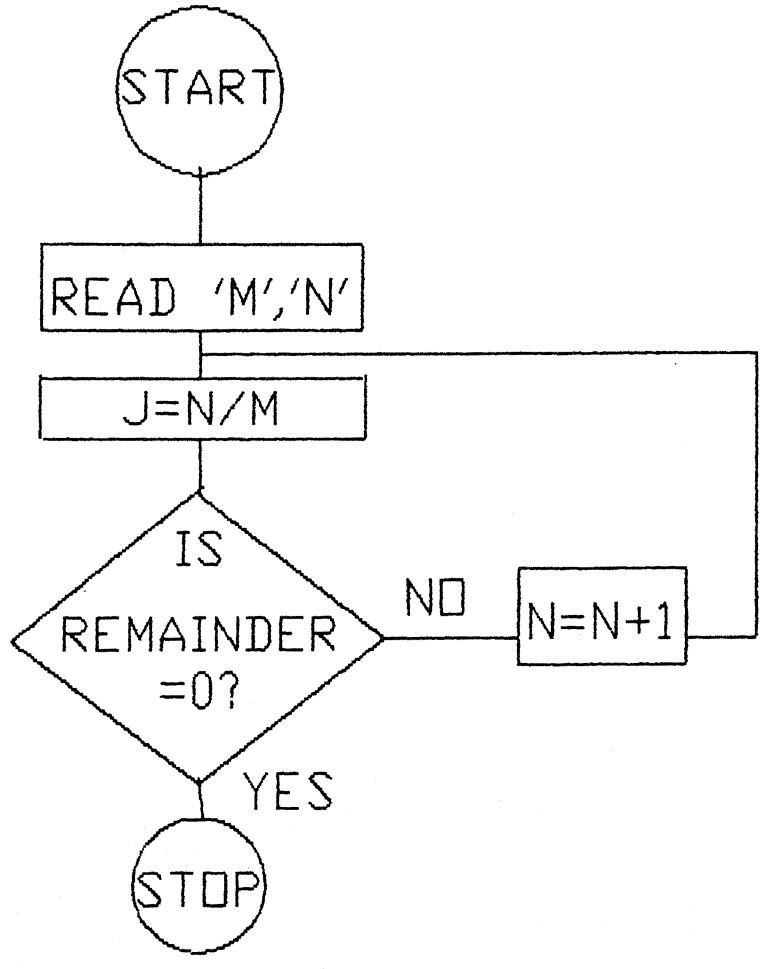


Fig. 4.4 : ZERO PADDING

4.3.4 Conversion of cartesian to polar form:

The data entry for X and Y by the user is in cartesian form only. For computations we need once to convert the cartesian to polar form. The flow chart for implementation of cartesian to polar form conversion of Y is given in Fig. 4.5. Converting the m-tuple of X into polar form say i.e. `i', the corresponding Y in polar form is stored in a buffer at the location j = 2i to arrange Y (polar form). The user has the option to enter all the X-Y data or quit the entry. In case of `quit' command, rest of the entries of Y may be taken as don't care. For computation time minimization the don't care entries are taken as all zero.

4.3.5 FFT computation over finite field:

To compute the `a' coefficients, DFT of $2^m-1$ data points is taken. In the software FFT of length $2^m-1$ has been implemented over appropriate finite fields. The implementation is based on Cooley-Tukey algorithm. Depending on `m' the FFT is taken over a length $N= 2^m-1$ data. The input data is Y (polar form). By Cooley-Tukey algorithm, we break N into K prime factors which are stored in a look up table. For prime length n, the n point Fourier transform problem is converted into n-1 point cyclic convolution using Rader prime algorithm. After the FFT realization, the `a' coefficients are generated according to Equation (3.5a) and

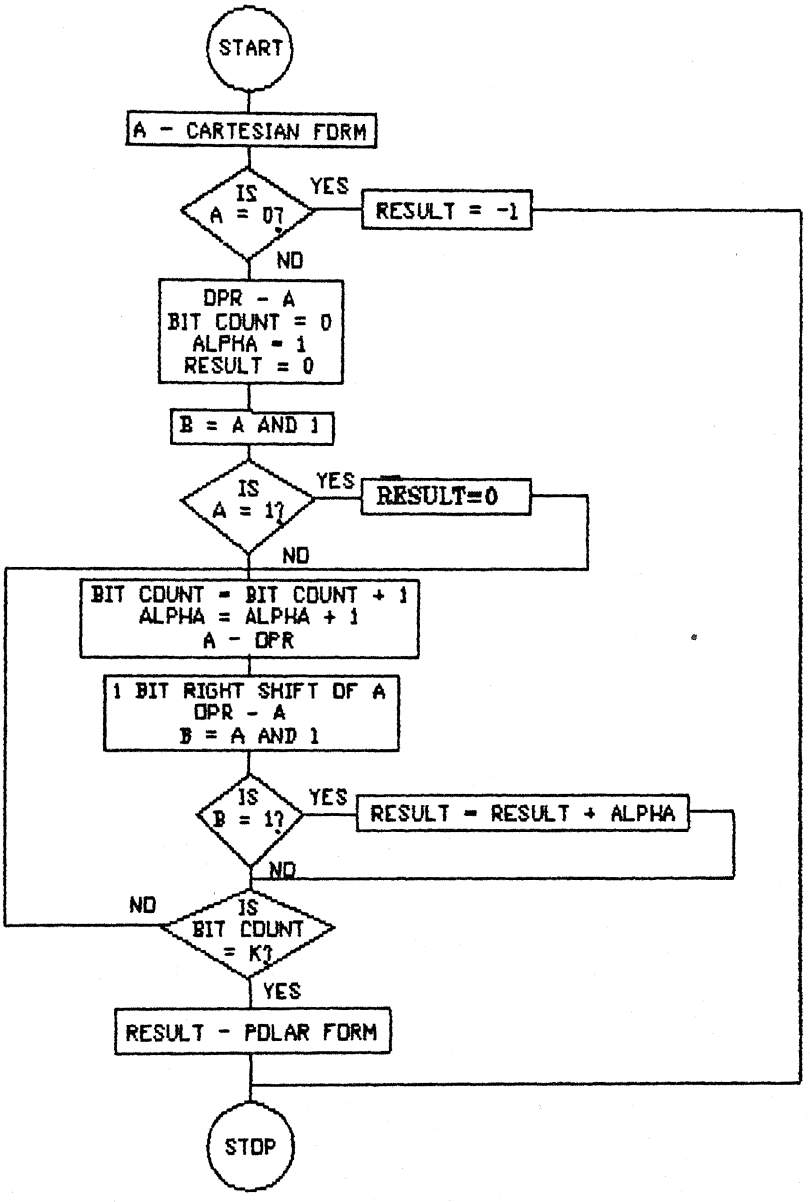


Fig. 4.5 : CONVERSION OF CARTESIAN TO POLAR

stored in the order of $a_{\beta^{-\infty}}$, $a_{\beta^{0}}$, $a_{\beta^{1}}$,.....,$a_{\beta^{2^m}-2}$

as discussed in Chapter 3.

In Chapter 5, we have presented the hardware implementation of polynomial evaluation over finite field. For this purpose, the `a' coefficients are arranged in the order of

$$a_{\beta^{-\infty}}\ ,\ a_{\beta^{2^m}-2},\ a_{\beta^{2^m}-3},\cdots\cdots,a_{\beta^{0}}$$

and stored in an output file "Coef.Dat" in cartesian form. From this output file, data are transferred to a EPROM for further use in hardware.

4.3.6 Polynomial evaluation to check the proper representation:

To check the polynomial representation user is asked to enter any X. With the computed `a' coefficients and X the polynomial is evaluated directly as given by Equation (3.2). The computed Y is in polar form. Conversion of polar to cartesian results the n-tuple of Y which is the same as entered by the user. Flowchart of polar to cartesian conversion is shown in Fig. 4.6. In this, the polar form is loaded in `Count' and `A' is an n bit shift register.

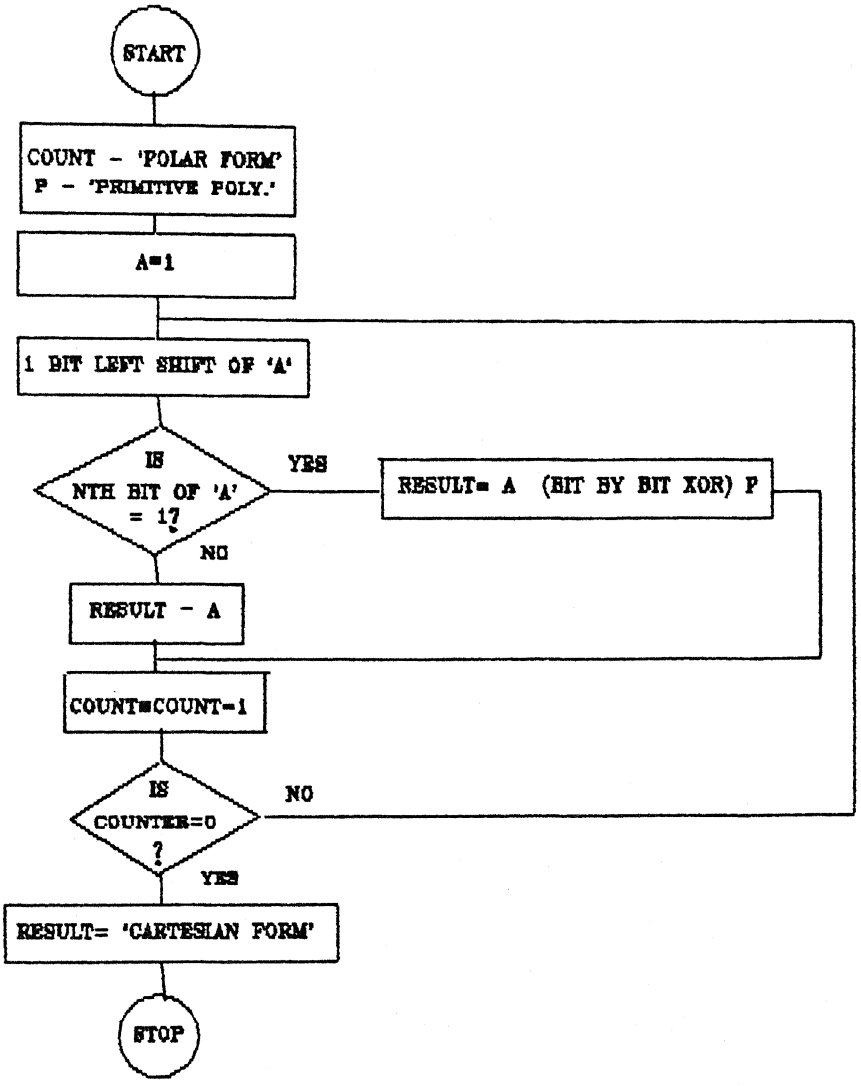


Fig. 4.6 : CONVERSION OF POLAR TO CARTESIAN

4.4 Realization of mapping for substitution network:

As an application of this polynomial representation, a mapping for substitution network has been realized. By this, we realize first the permutation of N objects and then this mapping of the original sequence to the permuted sequence is represented by a polynomial representation. Given a sequence of N objects there are N! possible permutations. With permutation number we can specify the permuted sequence. There is an unique mapping between the sequence and its permuted version for each permutation number. Permutation number `0000' means the permuted version sequence is the sequence itself. By the polynomial representation, permutation of N (2-8) objects have been realized. User has given the option either to enter the permutation table directly or permutation number. The maximum permutation number for N is N!-1 and is displayed on the screen to help the user to enter a valid permutation number.

The essential idea of generating permutation of N objects according to a permutation number is based on mixed radix representation known as `Factorial number system' [7]. In this system any integer number i : $0 \leq i \leq N!-1$ can be uniquely represented as

$$i = j_{N-1} * (N-1)! + j_{N-2} * (N-2)! + \ldots + j_1 * 1!$$

where $0 \leq j_{N-1} \leq N-1$, $0 \leq j_{N-2} \leq N-2$, ..., $0 \leq j_1 \leq 1$.

By a notation, it can be represented as: $i = < j_{N-1}\ j_{N-2} \cdots j_1 >$.

Using this representation we can generate the unique permutation of N objects i.e. { 1,2,3,...N} depending on `Permutation number'. The flow chart of the permutation generation is given in Fig. 4.7. The mapping of the sequence and its permuted version is then realized by polynomial representation and the corresponding `a' coefficients are displayed on the screen. Illustrative examples are shown in Fig. 4.8, Fig. 4.9, Fig. 4.10 and Fig. 4.11.

4.5 2-D Galois transformation:

From Equations (3.8) and (3.9), for m=2 we can represent a 2-variable function by a polynomial

$$y = f(x1,x2) = \sum_{i1=0}^{r} \sum_{i2=0}^{r} a_{i1i2} x^{i1} x2^{i2} \quad \text{.....(4.2)}$$

$$(x1, x2, y \in GF(2^n))$$

where $a_{00} = f(0,0)$

$$a_{i0} = \sum_{x \in GF(2^n)} x^{r-i} f(x,0)$$

$$a_{0i} = \sum x^{r-i} f(0,x) \qquad 1 \leq i \leq r \qquad \text{.....(4.3)}$$

and $$a_{i1i2} = \sum \sum x1^{r-i1} x2^{r-i2} f(x1,x2).$$

We can develop the following algorithm from the above Equations (4.3) to compute `a_{i1i2}' coefficients.

Algorithm `2-D Galois transformation'

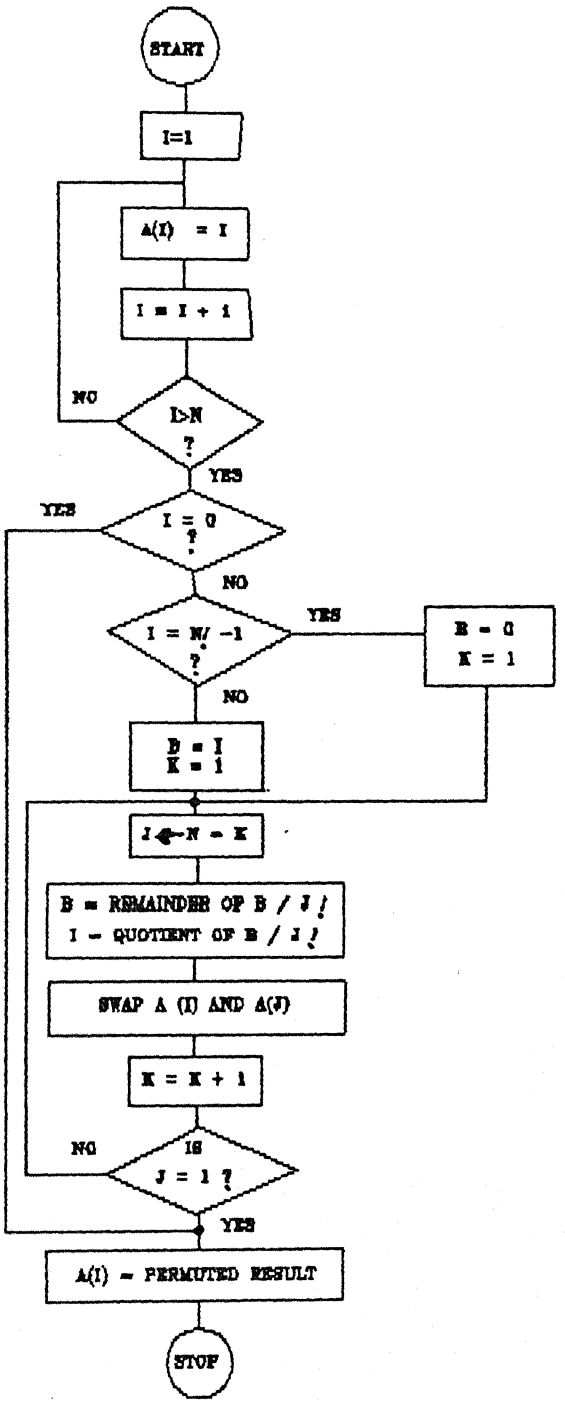


Fig. 4.7 : PERMUTATION GENERATION FROM PERMUTATION NUMBER

PERMUTATION TABLE

No. OF ELEMENTS= 07
PERMUTATION No.= 0000

X	Y
01	01
02	02
03	03
04	04
05	05
06	06
07	07

N.B: Y IS THE PERMUTED VERSION OF X

Fig. 4.8 : PERMUTATION TABLE (N=7, PERMUTATION No.=0000)

```
VALUE OF "a" COEFFICIENTS IN POLAR FORM
NOTE: all coefficients are power of primitive element(IN HEX).
Serial no. is left to the * & value of "a" coefs. to its right.
 @  REPRESENTS THE "0" FIELD ELEMENT IN POLAR FORM.          PAGE No.= 01

0000*    @
0001*    @
0002*    @
0003*    @
0004*    @
0005*    @
0006*    @
0007* 0000

        FIELD SIZE OF "a" Coefficients Is GF(2^03)
```

Fig. 4.9 : 'a' COEFFICIENTS FOR PERMUTATION TABLE WITH N=7

PERMUTATION TABLE No. OF ELEMENTS= 08

Y

PLEASE ENTER Y AT THE BRIGHT STAR POSITION

5
7
2 N.B: Y IS THE PERMUTED VERSION OF X
1
3
4
6
8

Fig. 4.10 : PERMUTATION TABLE FOR N=8 (DIRECT ENTRY)

VALUE OF "a" COEFFICIENTS IN POLAR FORM
NOTE: all coefficients are power of primitive element(IN HEX).
Serial no. is left to the * & value of "a" coefs. to its right.
@ REPRESENTS THE "0" FIELD ELEMENT IN POLAR FORM. PAGE No.= 01

```
0000*   @    000B* 0004
0001* 0003   000C* 0004
0002*   @    000D* 0003
0003* 000A   000E* 0005
0004* 000E   000F* 0009
0005* 0009
0006* 0005
0007* 0009
0008*   @
0009* 0008
000A* 0002
```

FIELD SIZE OF "a" Coefficients Is GF(2^04)

Fig. 4.11 : `a' COEFFICIENTS FOR PERMUTATION TABLE WITH N=8 (DIRECT ENTRY)

gin (2-D Galois transformation)

Set the value of N(5-7).

Read data from `Data file' to `Full_buff' of size 2^{2N}.

Store 1st 2^N consecutive data bytes in `Cart_buff'.

. Initialize Row_cnt = 1.

epeat:

. Compute 1-D Galois transform of the 2^N data in `Cart_buff'.

. Store back the transformed data to `Full_buff' in the locations xactly as read earlier.

. Row_cnt = Row_cnt + 1.

. Get next 2^N block of successive data from `Full_buff' to Cart_buff'.

ntil Row_cnt = 2^N.

. Initialize `STRT_ADDRESS' = 0 and `COL_NO' = 1.

epeat:

0. Scramble 2^N data from `Full_buff' as the starting location `STRT_ADDRESS' with offset of 2^N and store in `Cart_buff'.

1. Compute 1-D Galois transformation of the 2^N data in `Cart_buff'.

12. Store back the transformed data to `Full_buff' exactly in the locations as scrambled earlier.

13. `COL-NO' = `COL_NO' + 1.

14. `STRT_ADDRESS' = `Strt_ADDRESS' + 2^N.

Until `COL_NO' = 2^N.

. Open an output file `Imge.Dat' to store 2^{2N} transformed data.

. End { 2-D Galois transformation }.

For software implementation, input data of size 2^{2N} e taken from picture file of size $2^N \times 2^N$. The transformed data e stored in an output file `Imge.Dat'. To check the proper olynomial representation as given by (4.2), we have computed the orward Galois transform of input data followed by inverse Galois ransform. After these transformations output file contains the riginal input data to ensure the proper polynomial representation f 2-D function.

Advantage of such 2-D Galois transformation of 2^{2N} ata over 1-D transformation is that for 2-D case we need all the omputations over $GF(2^N)$. But for 1-D transformation of 2^{2N} data e need computations over $GF(2^{2N})$.

.6: Illustrative Examples:

For an arbitrarily chosen truth table, in the case f m = n = 5, the computed `a' coefficients and the entered truth able are shown in Fig. 4.12 and Fig. 4.13 respectively. To show ero padding another truth table example has chosen with m = 4 nd n = 5 with ten entries of X-Y. By three zero paddings to he n-tuple of Y we get the field size of `a' coefficients as $GF(2^8)$. Fig. 4.14 and Fig. 4.15 illustrate the example. For substitution mapping realization, we have given the example

```
                TRUTH TABLE
                ~~~~~~      No. Of X variables(m)= 05  No. Of Y variables(n)= 05
 NOTE: At blinking *,you have option for next entry or quit.
   For Entering next data use "RETURN", To quit use "ESC" key.
    X      Y        X      Y
*0000   0005    *000c   0010
*0001   0006    *0019   001e
*0002   001a
*0003   001d
*0004   0019
*0005   001d
*0006   0013
*0007   001c
*0008   001a
*0009   0011
*000a   000f
*000b   000d
```

Fig. 4.12 : TRUTH TABLE ENTERED BY USER (m=n=5)

VALUE OF "a" COEFFICIENTS IN POLAR FORM
NOTE: all coefficients are power of primitive element(IN HEX).
Serial no. is left to the * & value of "a" coefs. to its right.
@ REPRESENTS THE "0" FIELD ELEMENT IN POLAR FORM. PAGE No.= 01

```
0000* 0005  000B* 0016  0016* 0019
0001* 0003  000C* 000D  0017* 0008
0002* 0002  000D* 0003  0018* 001E
0003* 0003  000E*   @   0019* 0010
0004* 000D  000F* 0005  001A* 001B
0005* 0001  0010* 0017  001B* 0000
0006* 000F  0011* 000F  001C* 001D
0007* 0008  0012* 000F  001D* 0000
0008* 0017  0013* 0010  001E* 000F
0009* 000C  0014* 0001  001F* 0014
000A* 000A  0015* 0005
```

FIELD SIZE OF "a" Coefficients Is GF(2^05)

Fig. 4.13 : `a' COEFFICIENTS FOR TRUTH TABLE (m=n=5)

TRUTH TABLE

No. Of X variables(m)= 04 No. Of Y variables(n)= 05

NOTE: At blinking *,you have option for next entry or quit.
For Entering next data use "RETURN", To quit use "ESC" key.

X	Y	X	Y
*0000	000a	*000c	0017
*0001	0004	*000d	0018
*0002	0003	*000e	001a
*0003	000d	*000f	001f
*0004	000f		
*0005	0007		
*0006	0009		
*0007	000a		
*0008	000e		
*0009	000c		
*000a	0015		
*000b	0001		

Fig. 4.14 : TRUTH TABLE ENTERED BY USER (m=4,n=5)

VALUE OF "a" COEFFICIENTS IN POLAR FORM
NOTE: all coefficients are power of primitive element(IN HEX).
Serial no. is left to the * & value of "a" coefs. to its right.
@ REPRESENTS THE "0" FIELD ELEMENT IN POLAR FORM. PAGE No.= 01

```
0000* 0033   000B* 00DD
0001* 0081   000C* 00FA
0002* 0019   000D* 0016
0003* 0088   000E* 0042
0004* 0046   000F* 006A
0005* 00B6
0006* 0088
0007* 00C1
0008* 00F0
0009* 00C2
000A* 009E
```

FIELD SIZE OF "a" Coefficients Is GF(2^08)

Fig. 4.15 : `a'COEFFICIENTS FOR THE TRUTH TABLE (m=4,n=5)

for N = 7 with the permutation number as `0000'. Fig. 4.8 and Fig. 4.9 are the permutation table and corresponding `a' coefficients respectively. Another example is shown by Fig. 4.10 and Fig. 4.11 for N = 8 with the direct permutation table entry by the user.

The following table lists computation time of `a' coefficients for some combinations of m and n.

m	n	No. of entries	Computation time
8	8	10	2 secs
9	9	10	3.5 secs
12	12	10	8 secs
14	14	10	3 mins
15	15	10	5.5 mins

For 2-D Galois switching function realization we have chosen a picture file `Lincoln' of size 64 x 64 for forward as well as inverse galois transform. Forward transform followed by the inverse transform give back the original picture.

CHAPTER 5

HARDWARE IMPLEMENTATION OF POLYNOMIAL EVALUATION

5.1 Introduction:

This Chapter includes details of hardware mplementation. In Chapter 3 we have discussed the theory of olynomial representation of a system of Boolean functions. From uation (3.2), knowing the `a' coefficients the value of Y can be mputed corresponding to a given value of X. Using Horner's rule [6] r evaluating a polynomial expression we can reduce the number of ltiplications involved by (r-1) where r is the degree of polynomial. this rule the polynomial expression becomes :

$$y = f(x) = (\ldots((a_r x + a_{r-1})x + a_{r-2})x + \ldots)x + a_0).$$

sic functional schematic diagram for the realization of Horner's le is shown in Fig.5.2. Let us discuss the hardware implementation the following section.

2 Specification:

The hardware has been developed for $X, Y \in GF(2^N)$ th $N = (2\text{-}8)$. For a given mapping of X and Y, the `a' coefficients e loaded in a EPROM (IC 2716) from the output file "Coef.dat" eated by software. From the front panel N-tuple of X can be tered. For each N (2-8) the primitive polynomial with its N th bit

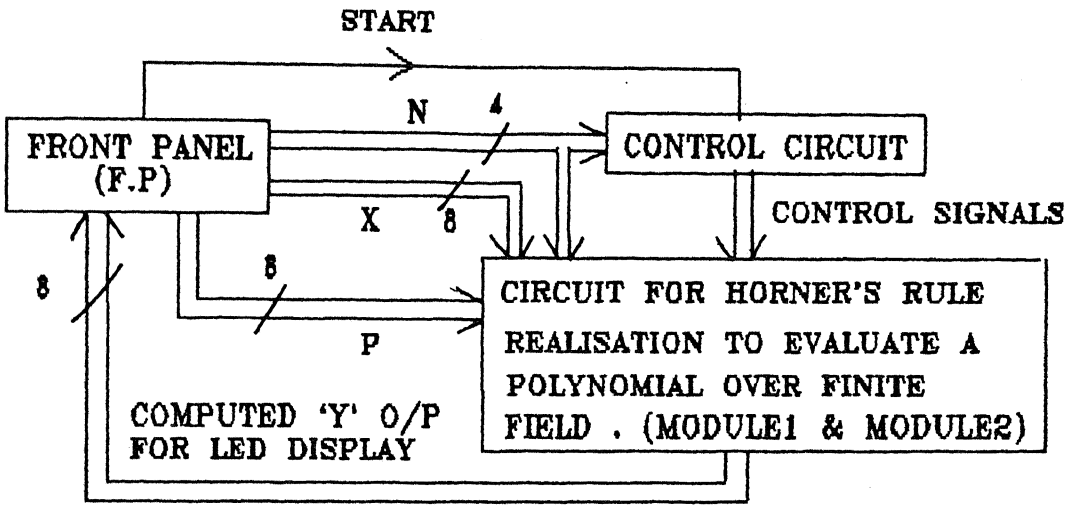


Fig. 5.1 : BASIC BLOCK DIAGRAM

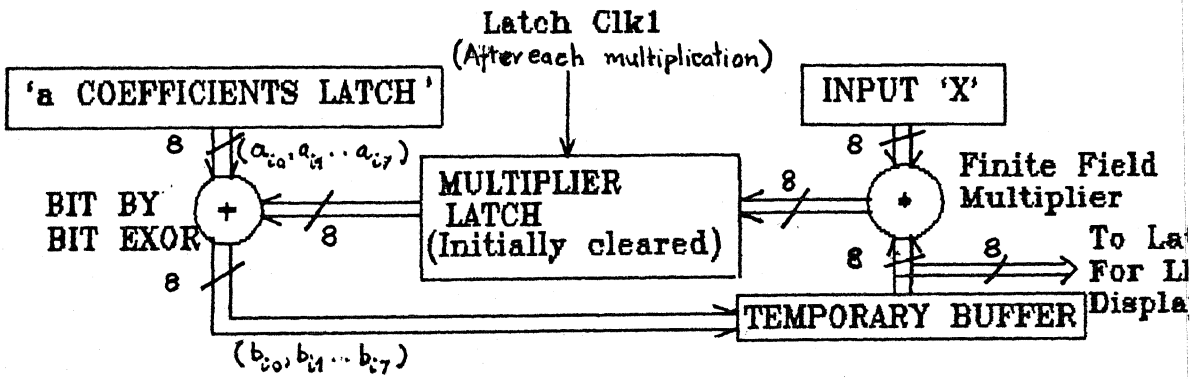


Fig. 5.2 : BASIC FUNCTIONAL SCHEMATIC DIAGRAM FOR HORNER'S RULE REALIZATION

as zero is also entered along with the entry of 4 digit bina representation of N. A start switch provided in the front pan initiates the computation and finally the computed output Y is latch to LEDs for display.

5.3 Hardware implementation based on Horner's rule:

Basic block diagram is shown in Fig. 5.1. From t front panel input X, number of variables N (2-8) and the primitiv polynomial of GF(2^N) are selected. The control circuit block as show basically generates all the control signals needed for Module1 an Module2. Module1 and Module2 together realize the entire circuit fo Horner's rule implementation.

5.3.1 Front panel:

The circuit diagram for the front panel has beer shown in Fig. 5.6. Switch `Sw' has been provided for starting th polynomial evaluation. This start information activates the control circuit to generate all the necessary timing signals. Fig. 5.7 depict the corresponding timing diagram for N=3. `SW-X' is used for selecting input X. The input data (X_0 X_1 X_2 . . . . X_7) are used by Module1. `SW-P' is to select the primitive polynomial (with N th bit as zero) for GF($2^N$). In Appendix B the primitive polynomial [5] for each N has been listed. `SW-N' is a 4 bit switch to enter the number of variables

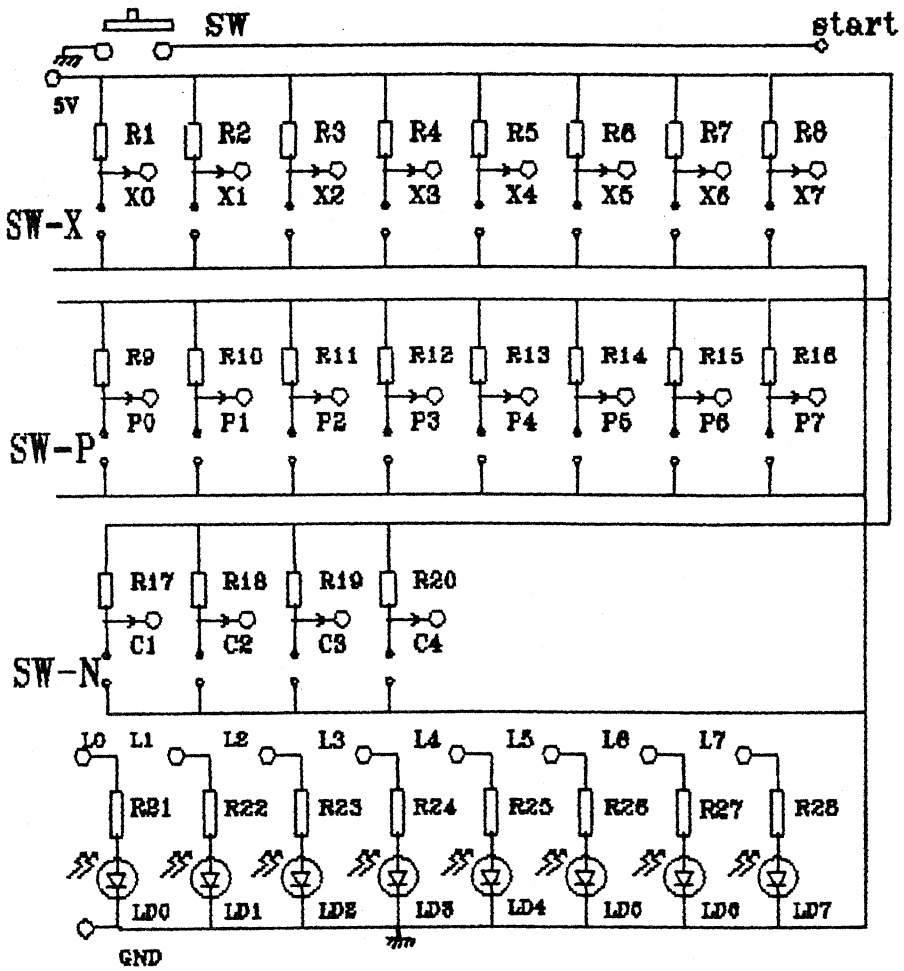


Fig. 5.6 - FRONT PANEL

COMPONENTS LIST: R1-R20: 1KOHM,1/4W, 5%

R21-R28: 270 OHM,1/4W, 5%.

SW-X,SW-P,SW-N: MICRO SWITCHES.

LD0-LD7: RED LEDS. SW-Push button Switch.

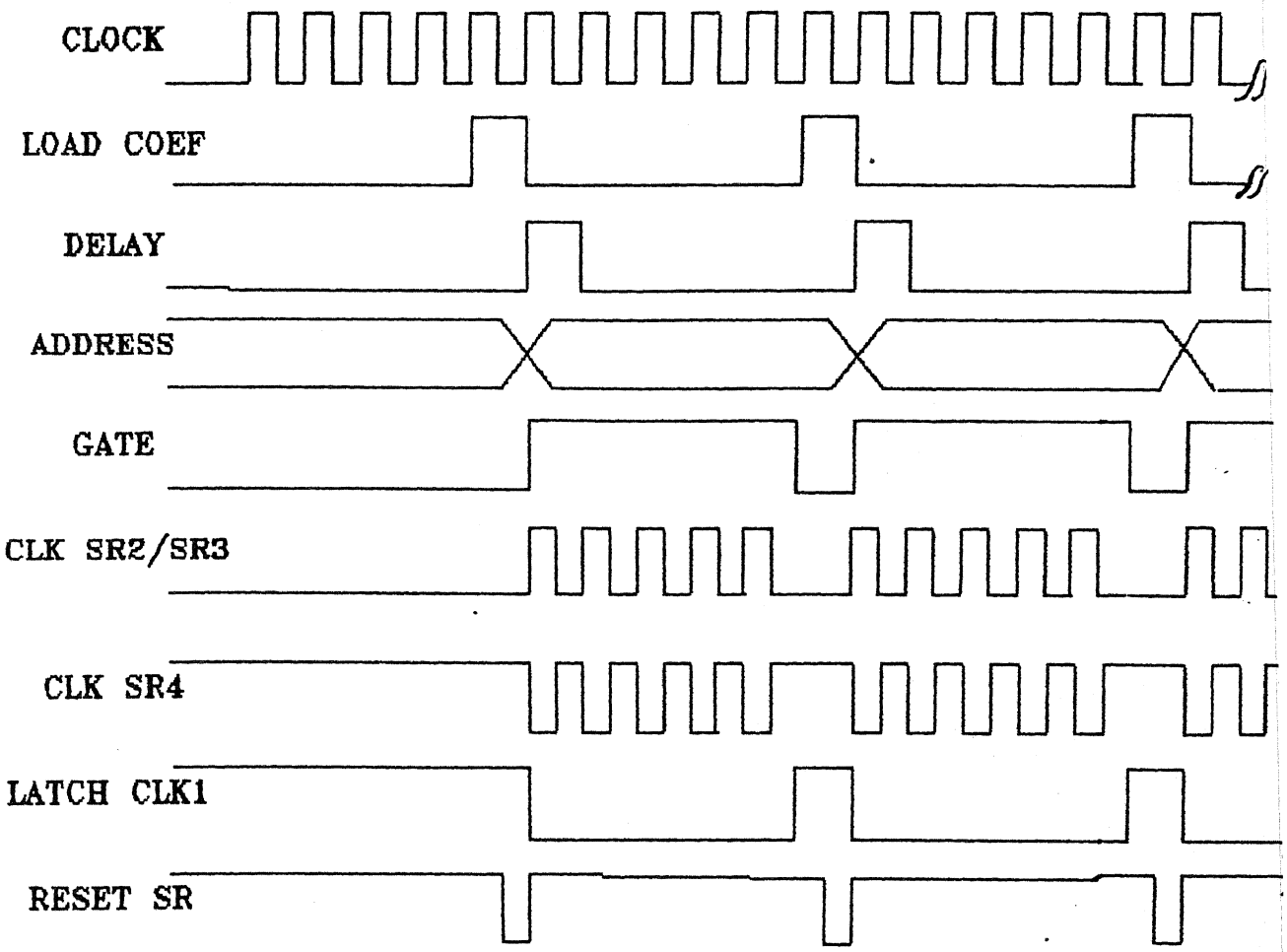


Fig. 5.7 : TIMING DIAGRAM (N=3)

N (2-8) as 4 digit binary number. Finally the red LEDs (LD0 - LD7 in the front panel display the computed Y output.

5.3.2 Control Circuit:

The circuit diagram is given in Fig. 5.3. With power on, a reset pulse is generated by R1, C1, D1 and U1 which resets U4, U9, U13, U11 and U14. As a result, the fundamental clock generated by R4, R5, C3, U2 and the crystal (4 MHz) is not allowed to pass through U5 . When the start switch is activated from the front panel, the output of the flip-flop (U4/Pin 11) becomes high and enables the clock to counter U5. From a 4 digit number N (2-8), U7 and U8 realize (2N - 1), which is necessary for the following purpose.

For multiplication of two field elements $\in$ GF(2^N) each N-tuple representation [5] of field element can be expressed as a polynomial of degree (N-1) with the coefficients from GF(2). Hence after multiplication of two such field elements can be expressed as a polynomial of degree (2N - 2). The polynomial has a total of (2N -1) coefficients including the constant term. With modulo operation of this result over a primitive polynomial of degree (N-1) we get the result of multiplication of two field elements as an N-tuple field element. For such multiplication we need (2N - 1) clocks. For this reason only we need to generate (2N - 1) from N. The multiplication is realized by Module1 and Module2 together.

U5, U6, U9 and U10 divide the input clock by

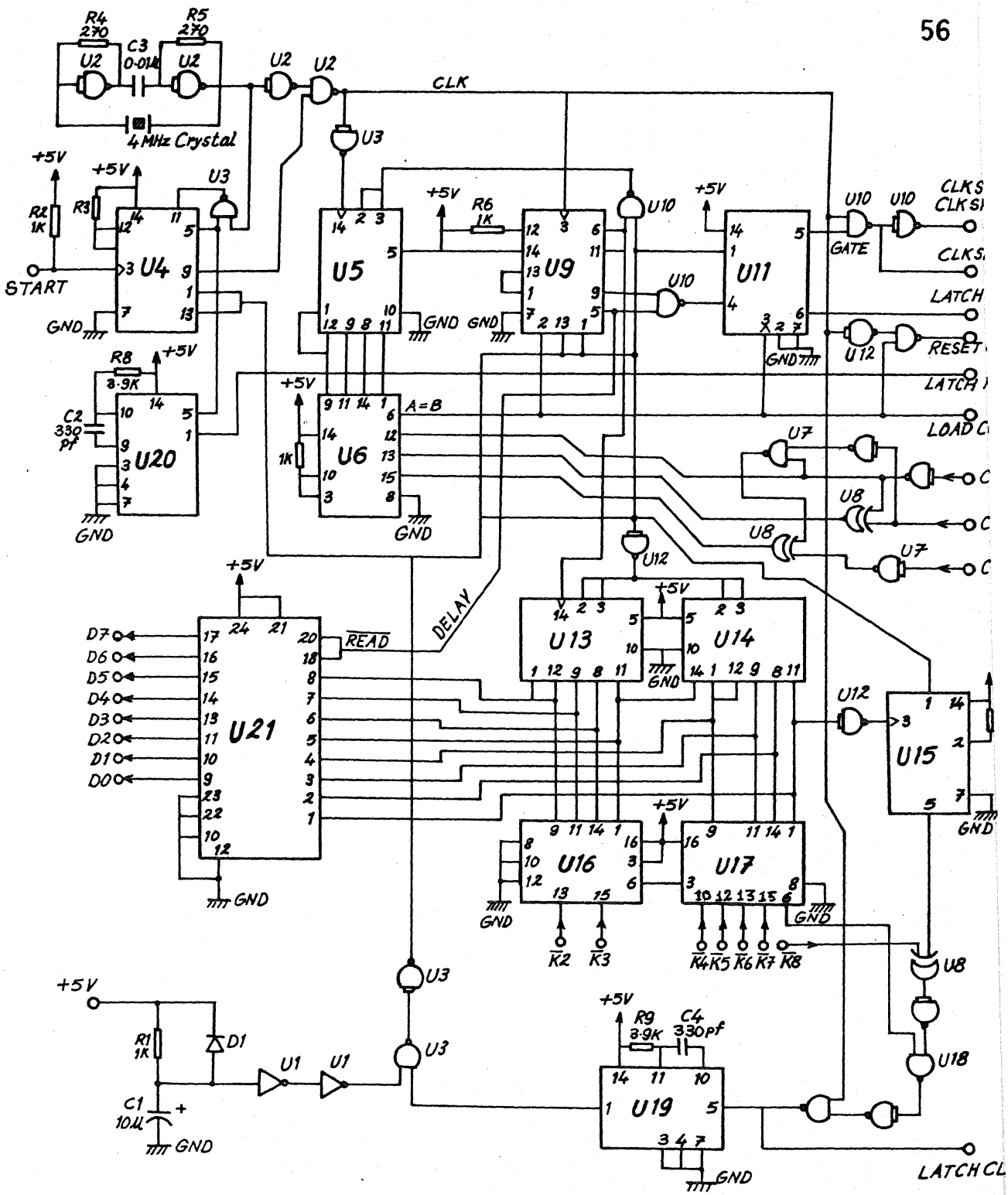


FIG. 5.3 CONTROL CIRCUIT DIAGRAM

Component list : U1 - 74LS13; U2 - 74S00; U3, U7, U10, U12, U18 - 74LS00; U8 - 74LS86; U4, U9, U11, U15 - 74LS74; U5, U13, U14 - 74LS93; U6, U16, U17 - 74LS85; U19, U20 - 74121; U21 - 2716.

All resistor are in ohms and capacitors in farad unless specified.

`(2N - 1)' to generate the signal `LOAD COEF'. This is delayed by one clock period to generate the signal 'DELAY' which is used as read signal for the EPROM U21. Counters U13 and U14 generate the 8 bit address for U21 starting from zero. At each rising edge of `DELAY' the address counter advances by one to fetch the next `a' coefficient. D flip-flop U11 is cleared at the rising edge of `LOAD COEF' and set at the rising edge of `DELAY' to generate the signal `GATE'. Fundamental clock is ANDed with `GATE' to generate `CLK SR2/SR3' to be used by Module1 and Module2 for multiplication purpose. The complement of this signal is `CLK SR4'. `RESET SR' is generated by NAND operation of `LOAD COEF' and `CLK'. For latching the multiplication result `LATCH CLK1' is used. After fetching all the 2^N `a' coefficients `LATCH CLK2' is generated to latch the final computed Y output and a monostable multivibrator U19 is triggered at the rising edge of `LATCH CLK2' to reset U4 to end the computation. The system then becomes ready for the next start to begin polynomial computation.

5.3.3 Module1 and Module2 for Horner's Rule Realization:

These two modules together realize Horner's rule implementation. Circuit diagrams for Module1 and Module2 are given in Fig. 5.4 and Fig. 5.5 respectively. In Module1 U21 is a 3-8 line decoder. Depending on C1, C2, C3 the outputs of decoder are given in the following table.

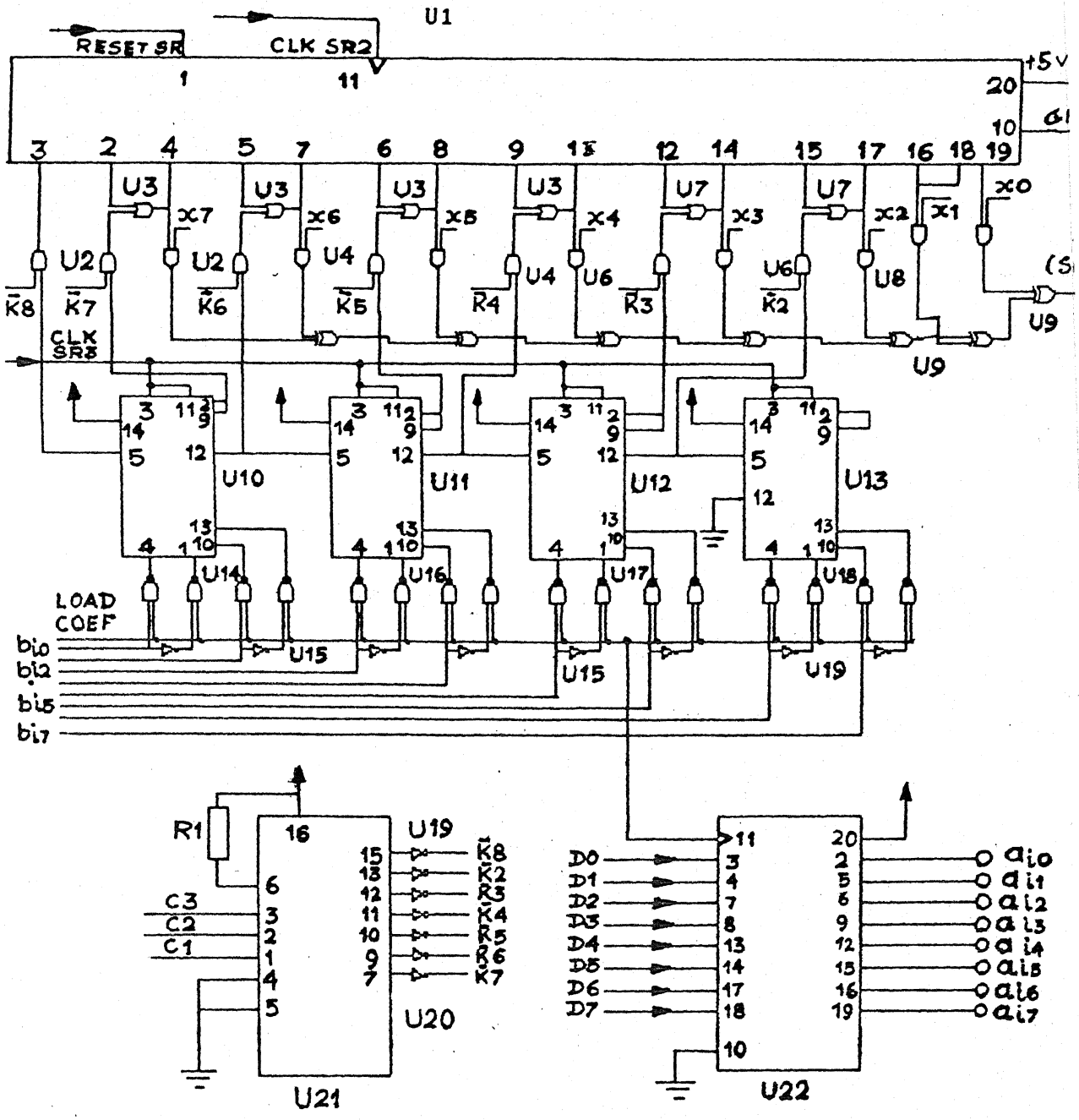
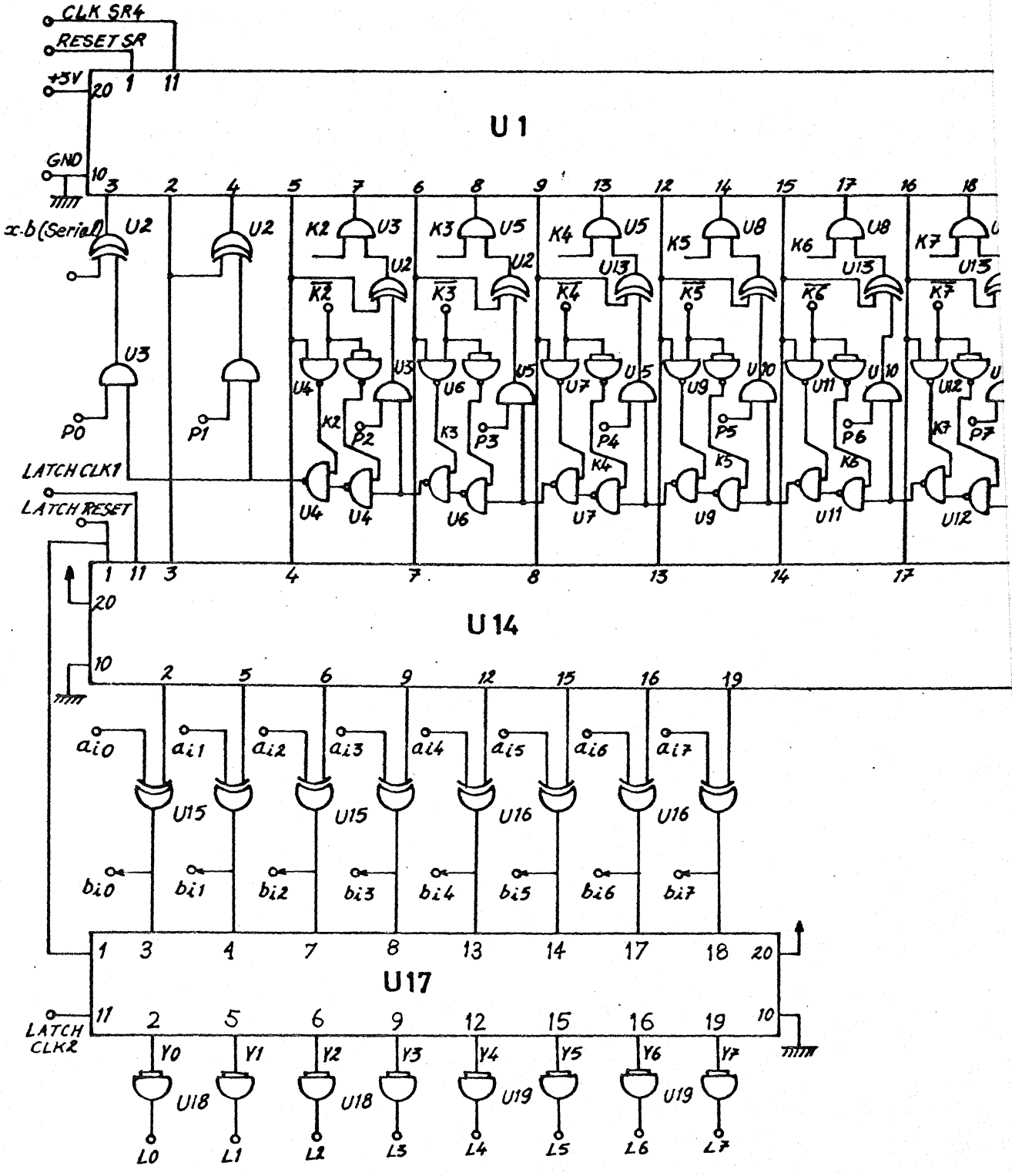


Fig. 5.4 : MODULE1 CIRCUIT

COMPONENTS LIST U1,U22-74273; U2,U4,U6,U8-74LS08;
U3,U7-74LS32; U5,U9-74LS86;
U10,U11,U12,U13 -74LS74; U14,U16-U18-74LS
U15,U19, U20-74LS04; U21-74LS138.
R1-1KΩ, 1/4W, + 5%.



TO THE LEDS OF FRONT PANEL

FIG-5.5 CIRCUIT DIAGRAM FOR MODULE-2

Components list : U1, U14, U17-74273; U2, U13, U15, U16-74LS86; U3, U5, U8, U10, U18, U19-74LS08; U4, U6, U7, U9, U11, U12-74LS00.

N	C4	C3	C2	C1	$\bar{K}2$	$\bar{K}3$	$\bar{K}4$	$\bar{K}5$	$\bar{K}6$	$\bar{K}7$	$\bar{K}8$
2	0	0	1	0	1	0	0	0	0	0	0
3	0	0	1	1	0	1	0	0	0	0	0
4	0	1	0	0	0	0	1	0	0	0	0
5	0	1	0	1	0	0	0	1	0	0	0
6	0	1	1	0	0	0	0	0	1	0	0
7	0	1	1	1	0	0	0	0	0	1	0
8	1	0	0	0	0	0	0	0	0	0	1

The data ($\bar{K}2$ $\bar{K}3$ $\bar{K}4$ $\bar{K}5$ $\bar{K}6$ $\bar{K}7$ $\bar{K}8$) are introduced to realize the generalized finite field multiplier over GF(2^N). For our problem N is in the range of (2-8). In the control circuit these data are used to compare the address count to 2^N to generate the signal `LATCH CLK2' for final latching of the computed output. In Module1 U22 latches the `a' coefficients at the rising edge of `LOAD COEF'. U10, U11, U12 and U13/Module1 are the `temporary buffer' (as in Fig. 5.2) which are transparent at the high level of `LOAD COEF'. U14/Module2 is the `Multiplier latch' (as in Fig. 5.2) which is cleared at each starting of the computation by the signal `LATCH RESET'. Bit by bit EXOR this multiplied result and the `a' coefficients is done by U15, U16/Module2. The EXOR output data (bi0, bi1, bi2, bi3, bi4, bi5, bi6, bi7) are transparently latched by `temporary buffer' for multiplication with X. U1, U2, U3, U4, U5, U6, U7, U8 and U9 of Module1 realize the multiplication of (bi0, bi1, bi2, bi3, bi4, bi5, bi6, bi7) and (X0, X1, X2, X3, X4, X5, X6, X7). With (2N - 1) clocks the (2N - 1) terms of the multiplication are transmitted serially in the sequence of MSB first and LSB last. This multiplier has been designed in generalized GF(2^N) with MSB as the 1st term in the serial output. With this MSB as the 1st term in the sequence the modulo

operation over this multiplied result is done in parallel with th multiplication operation. For multiplication each term of the resul is generated at the rising edge of `CLK SR2'. For the modulo operatio clock for the shift register U1/Module2 is just the complement o CLK SR2. After the elapse of (2N - 1) clocks the multiplication i over and the result is latched at the rising edge of `LOAD COEF'. Le us explain the field multiplication by the circuit with an example.

Example 5.1A : Suppose N=3. Primitive polynomial for GF(2^3) is taken as $x^3 + x + 1$. Let the field elements of GF(2^3) for multiplication be 110_2 and 101_2. For N=3, the equivalent circuit schematic diagram is shown in Fig. 5.8. For N=3 the number of clocks needed fo multiplication = 2N -1 = 2.3 -1 =5. The following table will explain clearly the circuit operation for multiplication.

Input b	Clk SR2 (↓)	Q2	Q1	Q0	x.b	CLK SR4 (↓)	$Q2^1$	$Q1^1$	$Q0^1$
-	No clock	0	0	0	0	No clock	0	0	0
1	↓	1	0	0	1	↓	0	0	1
1	↓	1	1	0	1	↓	0	1	1
0	↓	0	1	1	1	↓	1	1	1
0	↓	0	0	1	1	↓	1	0	0
0	↓	0	0	0	0	↓	0	1	1

After (2N - 1) clocks the multiplied result is 011_2.

The number of total clocks needed by the designed hardware to compute a polynomial of degree r = 2^n-1 are $N_c = 2n.(2^n-1)$.

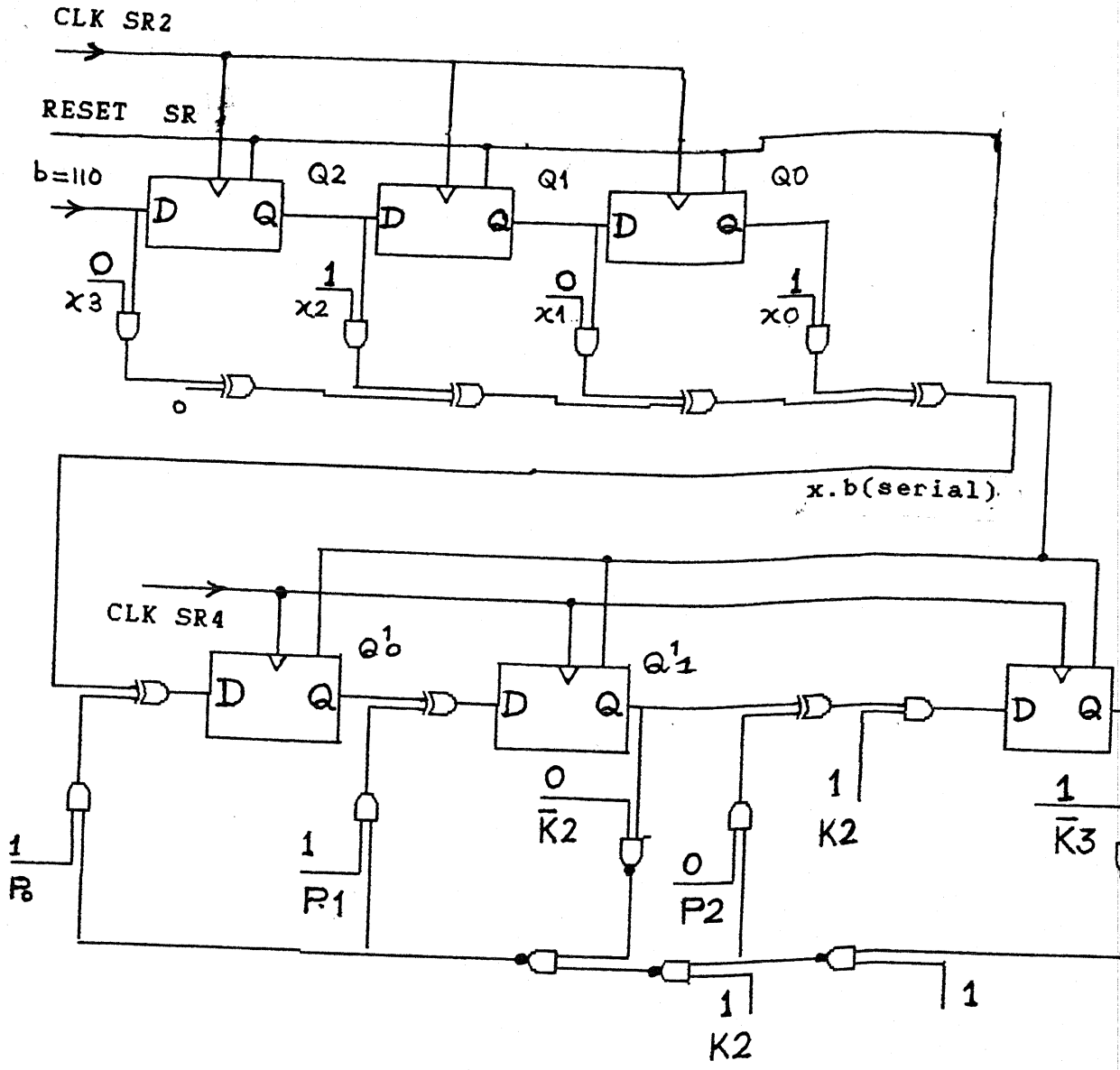


Fig. 5.8 : EQUIVALENT SCHEMATIC DIAGRAM FOR EXAMPLE-5.1A

5.3.4 Conclusion:

Recently Massey and Omura [12] developed a new multiplication algorithm for Galois fields based on normal basis representation. This is a very efficient multiplier and can be used effectively for a fixed Galois field $GF(2^m)$. Our approach is to develop a multiplier over $GF(2^n)$ with generalized N and the representation is based on standard basis. By Massey and Omura algorithm number of clocks for multiplication in $GF(2^m)$ = (m + 1). Our hardware implementation takes = (2m - 1) clocks for multiplication.

transformations, the transformed data should be identical to the input data.

Hardware implementation of polynomial evaluation over Galois field GF(2^N) with the range of N as 2 to 8 is based on Horner's rule. The multiplier design in standard basis is implemented in a modular structure so that it can be extended for higher N. VLSI implementation is possible by exploiting the modular structure.

Future scope:

For the computation of coefficients we have used Cooley-Tukey FFT algorithm. For prime lengths we have used the Rader prime algorithm to convert an n point discrete Fourier transform into an (n-1) point cyclic convolution. This convolution can be efficiently computed by using Winograd's convolution algorithm to improve the overall computation time.

Properties of Frobenius-cycles can be used to reduce the computation time for polynomial evaluation by a dedicated hardware. By the Frobenius transform theorem we have seen that if $a_i x^i$ is a term in the polynomial, then its Gf(q)-Frobenius transform $a_i^q x^{iq}$ can also be found in the polynomial. Thus the terms in the polynomial can be decomposed into several GF(q)-Frobenius cycles which can be computed in parallel to speed up the computation.

REFERENCES

1. Bartee, T.C., and Schneider, D.I.," Computation with finite fields. " Information and control 79-98, 1963.

2. Benjauthrit, B., and Reed, I.S., " Galois switching functions and their applications. " IEEE Trans. Comput. C-25, 78-86, 1976.

3. Benjauthrit, B., and Reed, I.S., " On the fundamental structure of Galois switching functions. " IEEE Trans. Comput. C-27, No. 8, 757-762, 1978.

4. Blahut, R.E., " Fast algorithms for digital signal processing. " New York: Addison-Wesley Publishing Co., Inc. 1983.

5. Blahut, R.E., " Theory and practice of error control codes." New York: Addison Wesley publishing Co.,Inc,1983

6. Doerr, A., and Levasseur, K.," Applied discrete structure for computer sceince.

7. Knuth, D., "Fundamental algorithms", the art of computer programming, vol.2.

8. Menger, K.S., Jr.," A Transform for logic networks." IEEE Trans. Comput. C-19, 132-140, 1969.

9. Ninomiya, I., " A theory of coordinate representation of switching functions. " Mem. Fac. Engrg, Nagoya Univ. 13, 149-363,1958.

10. Pradhan, D.K., and Patel, A.M., " Reed-Muller like canonic forms for multivariable functions." IEEE Trans. Comput.(Corresp.), 206-210, 1975.

11. Takahashi, I., "Switching functions construction by Galois extension fields." Information and Control 48, 95-108,1981.

12. Wang, C.C, Truong, T.K., Shao, H.M., Deutsch, L.j., Omura, J.K., and Reed, I.S., " VLSI architecture for computing multiplications and inverses in $GF(2^n)$." IEEE Trans. Comput., Vol. C-34, Aug.1985.

Appendix A

Appendix A : Proof of the theorem for a simple 1-D mapping:

The theorem states that any function f(x) on $GF(2^n)$ can be represented as a polynomial of order $r = 2^n - 1$;

$$f(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_r x^r$$

where, $a_i \in GF(2^n)$.

The coefficients a_i are determined by ;

$$a_0 = f(0)$$

$$a_i = \sum_{x \in GF(2^n)} x^{r-i} f(x), \qquad 1 \leq i \leq r.$$

Proof: We have, $f(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_r x^r \ldots . (A.1)$

Multipling both sides by x^{-i} we get;

$$f(x).x^{-i} = a_0 x^{-i} + a_1 x^{-i+1} + a_2 x^{-i+2} + \ldots . + a_i + \ldots . + a_r x^{r-i}.$$

Now, $x^r = 1$, $x \in GF(2^n)$ and $r = 2^n - 1$.

$$f(x).x^{r-i} = a_0 x^{r-i} + a_1 x^{r-i+1} + \ldots . + a_i + \ldots . + a_r x^{r-i}$$

We can write also;

$$\sum_{x \in GF(2^n)} f(x).x^{r-i} = a_0 \sum_{x \in GF(2^n)} x^{r-i} + a_1 \sum_{x \in GF(2^n)} x^{r-i+1} + \ldots . + a_i + \ldots + a_r \sum_{x \in GF(2^n)} x^{r-i}$$

We know for all $x \in GF(2^n)$, $\sum_{x \in GF(2^n)} x^j = 0$

Hence, $\sum_{x \in GF(2^n)} f(x).x^{r-i} = a_i, \qquad 1 \leq i \leq r.$

Now, put the value of x = 0 in Equation (A.1)

APPENDIX-B

Appendix B: List of primitive polynomials for $GF(2^k)$:

Here, we have listed primitive polynomials f($GF(2^k)$ for $2 \leq k \leq 15$.

k	Primitive polynomial	Hex representation (k th bit = 0)
2	$x^2 + x + 1$	3
3	$x^3 + x + 1$	3
4	$x^4 + x + 1$	3
5	$x^5 + x^2 + 1$	5
6	$x^6 + x + 1$	3
7	$x^7 + x^3 + 1$	9
8	$x^8 + x^4 + x^3 + x^2 + 1$	1d
9	$x^9 + x^4 + 1$	11
10	$x^{10} + x^3 + 1$	9
11	$x^{11} + x^2 + 1$	5
12	$x^{12} + x^6 + x^4 + x + 1$	53
13	$x^{13} + x^4 + x^3 + x + 1$	1b
14	$x^{14} + x^{10} + x^6 + x + 1$	443
15	$x^{15} + x^{12} + x^3 + x + 1$	100b

Date Slip

This book is to be returned on th date last stamped.
